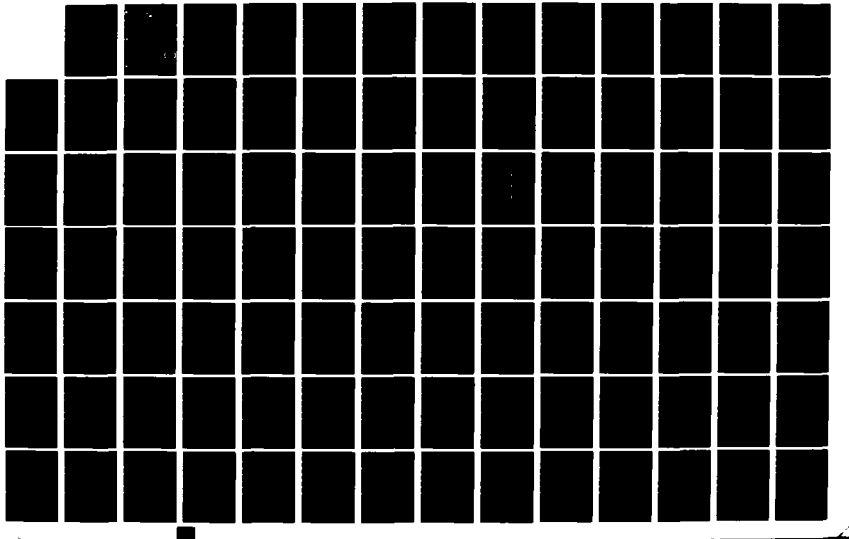
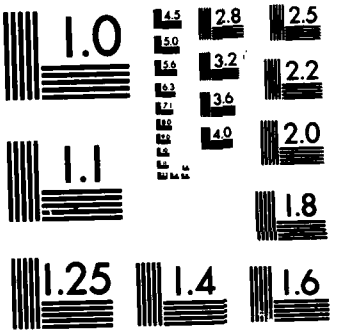
AD-A124 848 THE ANALYSIS AND DESIGN OF A COMPUTER SECURITY/RECOVERY 1/3
SYSTEM FOR A RELA.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W P RED
UNCLASSIFIED 17 DEC 82 AFIT/GCS/EE/82D-26 F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

THE ANALYSIS AND DESIGN OF A COMPUTER SECURITY/RECOVERY SYSTEM FOR A RELATIONAL DATABASE MANAGEMENT SYSTEM

THESIS

AFIT/GCS/EE/82D-26 William P. Reed
Capt USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY (ATC)
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFIT/GCS/EE/82 D-26	2. GOVT ACCESSION NO.: AD A124840
	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): THE ANALYSIS AND DESIGN OF A COMPUTER SECURITY/RECOVERY SYSTEM FOR A RELATIONAL DATABASE MANAGEMENT SYSTEM	5. TYPE OF REPORT & PERIOD COVERED:
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): William P. Reed, Captain USAF	8. CONTRACT OR GRANT NUMBER(s):
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB Ohio 45433	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
11. CONTROLLING OFFICE NAME AND ADDRESS:	12. REPORT DATE: 17 December 1982
	13. NUMBER OF PAGES: 228
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office):	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release: IAW AFR 190-17. 4 JAN 1983

LYNN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (ATC)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Security Model
Security
Relational Databases
Microcomputers

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A Computer Security/Recovery System (CS/RS) model was designed with the goal of implementing it into a Relational Database Management System. In addition, the relational database was to serve as a pedagogical tool for teaching database management and manipulation techniques as well as security measures and issues. Toward these goals, an extensive literature search and analysis of the current state of the art Computer Security models was required in order to identify current unresolved issues and areas

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AFIT/GCS/EE/82D-26

THE ANALYSIS AND DESIGN OF A COMPUTER SECURITY/RECOVERY SYSTEM FOR A RELATIONAL DATABASE MANAGEMENT SYSTEM

THESIS

AFIT/GCS/EE/82D-26 William P. Reed
Capt USAF

AFIT/GCS/EE/82D-26

THE ANALYSIS AND DESIGN OF A COMPUTER SECURITY/RECOVERY SYSTEM FOR A RELATIONAL DATABASE MANAGEMENT SYSTEM

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University (ATC)
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science

by

William P. Reed
Capt USAF

Graduate Information Systems

17 December 1982

PREFACE

This work presents a Computer Security model feasibility study, a requirements analysis for designing a Computer Security model, and the initial design of a Computer Security model for a Relational Database Management System. I hope this initial design effort will provide a sound basis for the actual implementation and testing which will need to be accomplished either as another thesis effort or a special study project.

I would like to express my deepest appreciation to Dr. Thomas C. Hartrum, who as my thesis advisor gave me guidance and encouragement. Thanks is also extended to Dr. Henry Potoczny and Major Michael Varrieur who as my thesis readers, provided constructive comments to improve the content and clarity of this thesis.

Finally, I wish to thank my wife, Sandra, and my daughters, Sonja and Carrie. They endured endless hours of complaining, however their patience, understanding, and selfless cooperation enabled me to complete this graduate program.

William P. Reed

CONTENTS

PREFACE ii

LIST OF FIGURES v

LIST OF TABLES vii

ABSTRACT viii

I. INTRODUCTION 1

BACKGROUND 1
STATEMENT OF PROBLEM 4
ASSUMPTIONS 4
SCOPE 6
GENERAL APPROACH 7
SEQUENCE OF PRESENTATION 7

II. SYSTEM CONSIDERATIONS 9

MACHINE 9
OPERATING SYSTEM 10
RELATIONAL DATABASE 11
SUMMARY 12

III. MODELS REVIEWED 14

ACCESS MATRIX MODEL 14
UCLA DATA SECURE UNIX MODEL 19
TAKE-GRANT MODEL 31
HIGH-WATER MARK MODEL 41
BELL AND LAPADULA (ORIGINAL) MODEL 44
BELL AND LAPADULA (REVISED) MODEL 57
SUMMARY 74

IV. MODEL REQUIREMENTS 77

GENERAL REQUIREMENTS 77
OPERATING SYSTEM REQUIREMENTS 78
USER REQUIREMENTS 78
DATA BASE ADMINISTRATOR REQUIREMENTS 79
RELATIONAL DATABASE REQUIREMENTS 80
MODEL SELECTION 81

V. CS/RS MODEL DESIGN 84

INTRODUCTION 84
CS/RS MODEL 84
CS/RS MODEL CHARACTERISTICS 86
RELATION AND DOMAIN CHARACTERISTICS 86
RELATION ACCESS RIGHTS 88

CS/RS MODEL COMPONENTS 89
DBMS ACCESS MONITOR 89
RELATION ACCESS CONTROL LIST 90
RELATION ACCESS MONITOR 94
CS/RS MODEL DATA STRUCTURES 95
DBMS ACCESS MONITOR 95
RELATION ACCESS CONTROL LIST 98
RELATION ACCESS MONITOR 101
CS/RS MODEL CONSTRAINTS 101

VI. CS/RS MODEL INTEGRATION AND TESTING 104

VII. CONCLUSION 109

OVERVIEW 109
FUTURE RECOMMENDATIONS 109
FINAL COMMENT 111

BIBLIOGRAPHY 112

APPENDIX A: USER'S GUIDE FOR THE AFIT RELATIONAL DATABASE SYSTEM -- THE DATA DEFINITION FACILITY 115

APPENDIX B: USER'S GUIDE FOR THE ROTH RELATIONAL DATABASE SYSTEM 145

APPENDIX C: CS/RS MODEL DATA FLOW DIAGRAMS 186

APPENDIX D: PUBLISHABLE ARTICLE 211

VITA 228

VOLUME II: PROGRAM LISTINGS AND DOCUMENTATION
(Available from AFIT/ENG)

LIST OF FIGURES

Figure		Page
1.	Portion of an Access Matrix	16
2.	UCLA Secure Unix Architecture	20
3.	Specification levels and consistency proofs .	24
4.	A de jure acquisition	32
5.	A de facto acquisition	33
6.	Comparison of properties of models	74
7.	Access Relation	96
8.	Temp_rel Relation	97
9.	Data Definition Facility structure chart . .	105
10.	Roth Relational Database structure chart . .	105
A-1.	The Data Definition Facility	118
B-1.	Roth's system without the DDL	149
C-1.	DBMGR Module Level 1.0	189
C-2.	ACL SETUP Module Level 1.3	190
C-3.	GET TREE Module Level 1.3.1	191
C-4.	DBMGR Options Level 1.4	192
C-5.	REMOVE RELATION Module Level 1.4.3	193
C-6.	ACT DELETE Module Level 1.4.3.2	194
C-7.	QUIT Module Level 1.4.4	195
C-8.	DBMAIN Module Level 1.0	196
C-9.	DB ACCESS MONITOR Module Level 1.1	197
C-10.	CHECK ID ACCESS Module Level 1.1.2	198
C-11.	DBMAIN Options Module Level 1.4	199
C-12.	UPDATE Module Level 1.4.1	200

Figure Page

C-13. ADD TO ACT Module Level 1.4.1.5 201

C-14. RETRIEVE USERID Module Level 1.4.1.6 202

C-15. SELECT OPTION Module Level 1.4.1.6.4 203

C-16. EDIT Options Module Level 1.4.2 204

C-17. VALID INSERT COMMAND Module Level 1.4.2.1 . 205

C-18. VALID DELETE COMMAND Module Level 1.4.2.2 . 206

C-19. VALID MODIFY COMMAND Module Level 1.4.2.3 . 207

C-20. RETRIEVE Option Module Level 1.4.6 208

C-21. EXECUTE Module Level 1.4.6.4 209

C-22. DISPLAY Module Level 1.4.6.5 210

D-1. Comparison of properties of models 217

D-2. Data Definition Facility structure chart . . 221

D-3. Roth Relational Database structure chart . . 222

LIST OF TABLES

Table		Page
I	Classes of Requests	65

ABSTRACT

A Computer Security/Recovery System (CS/RS) model was designed with the goal of implementing it into a Relational Database Management System. In addition, the relational database was to serve as a pedagogical tool for teaching database management and manipulation techniques as well as security measures and issues.

Toward these goals, an extensive literature search and analysis of the current state of the art Computer Security models was required in order to identify current unresolved issues and areas of research. The advantages and disadvantages associated with six computer security models were explored and addressed in this thesis. Then the requirements for a CS/RS model were identified and defined, and based on certain favorable criteria, one of the computer security models analyzed was selected.

Further, only one of the six models had ever been implemented using a relational database and none had ever been implemented using a microcomputer. Consequently, the ability to take one of the existing computer security models and to apply the security concepts used within this model was not an easy task and in some instances was not possible.

Finally, the CS/RS model designed did not include the hierarchically ordered military security sensitivity levels.

THE ANALYSIS AND DESIGN
OF A
COMPUTER SECURITY/RECOVERY SYSTEM
FOR A
RELATIONAL DATABASE MANAGEMENT SYSTEM

I INTRODUCTION

BACKGROUND

The rising abuse rate of computers and the increasing threat to personal privacy through databases have stimulated an increased interest in the technical safeguards for data. Efforts to build secure computer systems have been underway for more than a decade. Many designs have been proposed, some prototypes have been constructed, and a few systems are approaching the production stage. A small number of systems in the Department of Defense (DOD) are even operating in "multilevel" mode: some information in any of these systems may have a classification higher than the clearance of some users (Ref 1:247).

Nearly all of the projects to design or construct secure systems for processing information have had a formal mathematical model for security as part of the top-down design definition of the system. The model functions as a concise and precise description of the behavior desired of the security-relevant portions of the system.

In order to build a secure system, designers need to first decide exactly what "secure" means for their

particular needs. For instance, in a private company, security may be related to nondisclosure of confidential accounting data or trade secrets, or to the enforcement of privacy regulations regarding personal medical or credit records. On the other hand, if national security data are involved, security becomes the protection of classified material as outlined in various DOD instructions and regulations. In either case computer security encompasses the measures required to (1) protect a computer-based system, including its physical hardware, personnel, and data against deliberate or accidental damage; (2) protect the system against denial of use by its rightful owners; and (3) protect information or data against divulgence to unauthorized recipients (Ref 2:55).

Most of the time it is evident that different security measures are required when limiting access to programs and data in a computer system having multiple users. In many systems where multiple users do exist there may be applications which require that not all users have the same idential authority. When this is true, it might be necessary to devise some scheme to ensure that the computer system prevents any unauthorized access to the data. For example, an instructor who maintains his course grades on an interactive system may wish to allow each student to read his own grades but not the grades of other students. He may also wish to allow any student to examine a histogram of the class grades for any assignment. In

addition, the instructor would like an assistant to enter new grades, but he wants to guarantee that each grade entered cannot be changed later without the instructor's specific approval.

Many other examples of systems requiring protection of information are encountered every day: credit bureau data banks; law enforcement information systems; time-sharing service bureaus; on-line medical information systems; and government social service data processing systems (Ref 2:67). These examples span a wide range of needs for organizational and personal privacy. All have in common controlled sharing of information among multiple users. All, therefore, require some type of a security plan to ensure that unauthorized access, modification and/or deletion of data does not occur.

In regards to the Air Force Institute of Technology (AFIT) Digital Engineering Laboratory (DEL), most of the available computer operating systems do not address all areas of security. For instance, the security measures established by the DEL for the Digital Equipment Corporation (DEC) LSI-11 microprocessor configuration are quite lenient. As a result, any individual can access any one of the several operating system diskettes available for use on the LSI-11s. Each diskette has a system directory where the names of the files that have been created through the system are maintained. In addition, each operating system allows any user to observe its directory of file

names. Consequently, any individual has the capability to delete, modify, add, or rename any file within the operating system's directory. It is quite obvious that potential security problems may arise in this area.

STATEMENT OF PROBLEM

The problem addressed in this thesis is divided into three subtasks: first, an extensive literature search of the current state of the art Computer Security/Recovery Systems (CS/RSs) is required in order to identify current unresolved issues and areas of research; second, the requirements necessary for the design of a CS/RS model are identified and defined; and finally, a CS/RS model is designed for a microprocessor using a Relational Database Management System.

ASSUMPTIONS

1. Several of the computer security models researched emphasized that most security measures should be implemented at the operating system level. However, due to the constraints imposed by the DEL DEC LSI-11 microprocessor configuration, security measures pertaining to the operating systems, printers, diskettes, files, paper, and other peripherals are not addressed in this thesis.

2. The concept of backup and recovery is very important, especially when a computer database is constantly being modified or updated. In most cases, the

operating system or computer operators implement any backup and recovery procedures automatically without user intervention. However, due to the nature of the LSI-11 configuration it is almost impossible to implement backup and recovery procedures without some type of interaction by the Data Base Administrator (DBA) or the user. As a result, only suggestions (Chapter V) for backup and recovery procedures are addressed in this thesis.

3. The CS/RS model was assumed to operate within a single level, single user environment. This assumption was necessary because simultaneous, multi-user access was not permitted due to the constraints imposed by the Data Base Management System (DBMS) and the operating system.

4. The fourth assumption dealth with the concept of integrity (the prevention of errors by users due to their carelessness or lack of knowledge). For example, let some tuple attribute within a relation be modified by a user. In addition, this same tuple attribute exists within a second relation but does not get modified for some reason. Now the integrity of the relational database is in jeopardy because this same tuple attribute assumes two distinct values. This problem, however, was not addressed in the CS/RS model designed because of the thesis time constraints.

5. Finally, the assumption was made that any temporary relations created by any transactions would not be visually observable to the Data Base Administrator (DBA) or the users. In addition, after the completion of any

transactions, all temporary relations created would be deleted. This assumption was necessary in order to prevent any unauthorized accesses to data at the temporary relation level.

SCOPE

The major scope of this thesis consisted of an extensive literature search on each of the following computer security models and their associated advantages and disadvantages: (1) access matrix; (2) the University of California, Los Angeles (UCLA) Data Secure Unix; (3) take-grant; (4) high-water mark; (5) Bell and LaPadula (original); (6) Bell and LaPadula (revised); and (7) others as encountered. The literature search included but did not specifically focus on security measures in the following computer areas: (1) procedural; (2) personnel; (3) physical; (4) hardware; and (5) software.

The CS/RS model design was based on several of the above computer security models. In addition, any security measures pertaining to the LSI-11 operating systems and their peripherals was not discussed or implemented in the CS/RS model designed. Also, only suggestions for backup and recovery procedures are mentioned.

The procedures are given for the integration and testing of the CS/RS on the LSI-11 microprocessor using the Roth Relational Database System (Ref 6) and the UCSD Pascal Operating System (Ref 5).

GENERAL APPROACH

The first step consisted of an extensive literature search on each of the previously mentioned security models. An effort was made to collect information on the advantages and disadvantages associated with each security model. These are presented in Chapter III.

In order that a direction be established for the rest of the thesis, the next step required defining the requirements that were necessary to adequately identify the security measures that would be implemented on the LSI-11 microprocessor using the Roth Relational Database.

Once these two steps were accomplished, a CS/RS model was designed to interface with the LSI-11 microprocessor using the Roth Relational Database. This required designing and coding several computer programs.

After this step was completed, procedures for the integration and testing of the CS/RS model were discussed. This was necessary to ensure that the model as designed was actually being implemented.

Finally, conclusions were drawn based on the designed CS/RS model; problems encountered while designing and coding the model were identified; and recommendations for future studies or thesis topics were made.

SEQUENCE OF PRESENTATION

The remainder of this thesis is broken into six chapters. Chapter II describes the computer which was used

to implement the CS/RS model. Machine size and availablity, the operating system features and two relational databases are also discussed. Chapter III discusses the various computer security models and the advantages and disadvantages associated with each model.

Chapter IV is a description of the requirements for the design of the CS/RS model. Chapter V describes the design of the CS/RS model based on those requirements. Chapter VI describes the procedures for integrating and testing the CS/RS model designed and Chapter VII presents an overview, conclusions, and recommendations.

II SYSTEM CONSIDERATIONS

This chapter describes the computer system used in implementing the CS/RS model. It addresses machine size and availability, the operating system, and two relational database systems.

MACHINE

One of the primary goals was to implement the CS/RS model on the Digital Equipment Corporation (DEC) LSI-11 microprocessor.

The LSI-11 is a 16-bit microcomputer with the speed and instruction set of a minicomputer. Due to its size and unique capabilities, it can fit into almost any instrumentation, data processing or controller configuration. Some of its more noteable features include: a 400 plus instruction set; extensive computer power; small processor size; modularity; serial and parallel I/O modules; add-on memory; stack processing; and Direct Memory Access (DMA).

Currently, there are five LSI-11s (four more are on order) which are interfaced in a nonsymmetric network (Ref 3). Eventually, the LSI-11 network will be linked into the overall Digital Engineering Laboratory Network (DELNET).

Since the LSI-11s provide portability of data (relations in respect to this thesis) and programs between machines, a very flexible network interface is available. Consequently, the CS/RS model designed and implemented

would operate on any one of the LSI-11s without any modifications to the source code or the microprocessors.

OPERATING SYSTEM

One of the ways that machine communication was simplified was through the use of the UCSD (University of California, San Diego) Pascal Operating System, and a high level implementation language, namely UCSD PASCAL, Version II.0 (Ref 5).

The UCSD Pascal Operating System was chosen because it can be run on stand alone micro/minicomputers. In addition, the system is highly machine independent since it runs on a pseudo-machine interpreter commonly referred to as the P-machine. All the system software is written in PASCAL, except for the P-machine interpreter and several run-time support routines written in assembler for efficiency, resulting in relatively straightforward software maintenance and enhancement (Ref 5:1). The Pascal System also includes a screen oriented editor specifically designed for use on Video Display Terminals.

PASCAL was chosen as the implementation language for several reasons. Most important is the block structured design of the language, allowing a smooth transition from a top-down design to the actual code (Ref 6:9).

Other reasons for choosing PASCAL are its high degree of variable typing and its wealth of control structures. The UCSD version of PASCAL is a widely used implementation language among microcomputers, thus providing a more

portable system. UCSD PASCAL was also chosen because, by being designed for microcomputers, it has a built-in capability for handling large programs with features such as program segmentation and separately compiled procedures and virtual memory using segment swapping. Since UCSD PASCAL normally runs in 48k bytes of main memory, the 54k bytes of main memory available on each LSI-11 is quite adequate. It is also important to note that both random and sequential access to dual disk drives are available in UCSD Pascal. Thus, the database system may reside on one disk so that procedures may be swapped in and out of memory, and user data (relations) may reside on the other disk. By simply switching user disks, data space becomes virtually unlimited (Ref 6:9).

RELATIONAL DATABASE

The CS/RS Model was to be implemented on either the Digital Engineering Laboratory (DEL) Backend Database System (Ref 26) or the Roth Relational Database System (Ref 6).

The DEL Backend Database System was described by Robert W. Fonden in a December 1981 thesis (Ref 26). Fonden's primary goal was to implement efficient and optimal data storage and retrieval techniques on a specialized backend database architecture. The model he implemented called for a host computer that would send queries to a Backend Control Processor (BCP) which

controlled several parallel Query Processors (QP). The BCP would assign certain queries to each QP, and download pages of relations from the main database to an Intermediate Memory Module (IMM) at each QP. This would allow several QPs to process a query in parallel on different pages of a relation.

The Roth Relational Database, was described by Mark A. Roth in a December 1979 thesis (Ref 6). This database was to be implemented as a general purpose system with specific provisions as a pedagogical tool for teaching database management and manipulation. Techniques were incorporated into the database that would transform user inputs by optimization for execution. This approach was intended to minimize query response time and space utilization. Although a large portion of this relational database is still to be implemented, a baseline model is on-line and operational.

The major reason for choosing the Roth Relational Database over the DEL Backend Database System was due to a better overall design which appears to have a higher probability of actually being implemented. Also since the Roth Relational Database was to be used as a pedagogical tool for teaching database management and manipulation techniques it would accomplish more immediate short term objectives.

SUMMARY

The combination of a dedicated microprocessor (the

LSI-11), the versatility of the UCSD Pascal operating system and language, and the Roth Relational Database proved very beneficial in the design and implementaion of the CS/RS model.

III Models Reviewed

This chapter describes the following computer security models and their associated advantages and disadvantages: (1) Access Matrix; (2) UCLA Data Secure Unix; (3) Take-grant; (4) High-water Mark; (5) Bell and LaPadula (original); (6) Bell and LaPadula (revised); and (7) Others.

ACCESS MATRIX MODEL

The access matrix model for computer protection is based more on abstraction of operating system structures than on military security concepts (Ref 1:256). One of the earliest descriptions of this model was provided by Lampson (Ref 7).

There are three principal components in the access matrix model: a set of objects, often called X; a set of domains, often called D; and an access matrix, often called A. Objects are the things which have to be protected. Typically, objects consist of files, terminals, processes, domains, segments and other entities implemented by an operating system. Domains are the entities which have access to objects. The essential property of a domain is that it has potentially different accesses than other domains. It is important to note that every domain can be an object, thus it may be read or otherwise manipulated by another domain.

The access of domains to objects is determined by the

access matrix. Its rows are labeled by domain names and its columns by object names. Element A[i,j] specifies the access which domain i has to object j. Each element for a particular row and column consists of a set of strings called access attributes. These attributes reflect the mode of access between the corresponding domain and object. Typical access attributes are read, write, append, control, and execute. In addition, a copy flag may be used to record ownership of a particular object.

Entries in the access matrix are made and deleted according to certain rules. Refer to the access matrix in Figure 1. A domain d1 can modify the list of access attributes for domain d2 and object X as follows:

a) d1 can remove access attributes from A[d2,X] if it has 'control' access to d2. Example: d1 can remove attributes from rows 1 and 2.

b) d1 can copy to A[d2,X] any access attribute it has for X which has the copy flag set and can say whether the copied attribute shall have the copy flag set or not. Example: d1 can copy 'write' to A[d2,file 1].

c) d1 can add any access attribute to A[d2,X] with or without the copy flag, if it has 'owner' access to X. Example: d2 can add 'write' to A[d2,file 2].

d) d1 can remove access attributes from A[d2,X] if d1 has 'owner' access to X, provided d2 does not have 'protected' access to X. The 'protected' restriction allows one 'owner' to defend his access rights from other 'owners' (Ref 7:21).

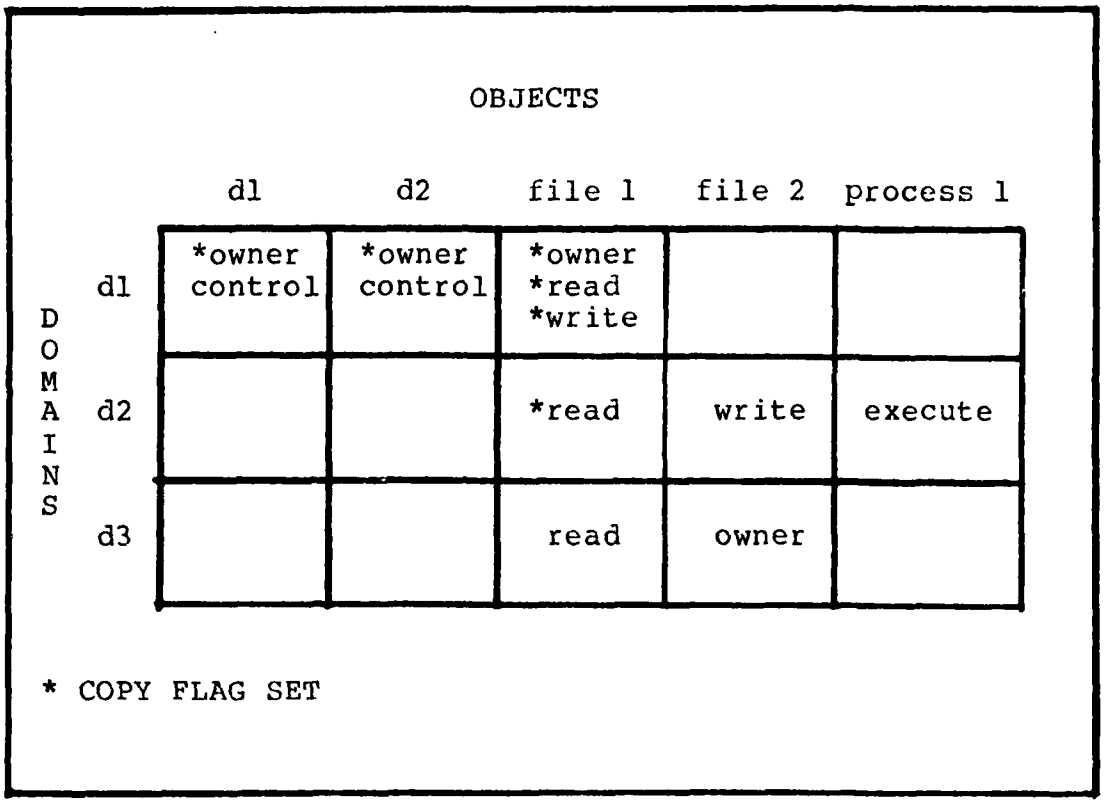


Figure 1. Portion of an Access Matrix (Ref 7:22).

The access matrix can be viewed as recording the protection state of the system. All accesses to objects by domains are assumed to pass through an enforcement mechanism that refers to the data in the access matrix. This mechanism, called a reference monitor, rejects any accesses (including improper attempts to alter the access matrix data) that are not allowed by the current protection state and rules (Ref 8).

DISADVANTAGES

In actual computer systems, the access matrix would be very sparse if it were implemented as a two-dimensional

array. This dilemma can be handled in three ways. The first would be to store the access matrix as a table of triples (D,X,A[D,X]). Since most domains and objects would not be active at any given time, it would be unnecessary to store all the triples in fast access memory at once. This solution creates a new problem. Suppose it is necessary to be able to determine what object a given domain can access or vice versa. A table of triples would have too little structure to permit efficient search procedures since it might be necessary to search the whole table. The second solution would be to store the access matrix by rows. Now each domain D has associated with it a list of pairs (X,A[D,X]), each pair being called a capability and the list a capability list (C-list) (Ref 9). A C-list represents all the objects that a domain can access. Because the protection system allows only authorized operations on C-lists, possession of capability by a process is prima facie proof that the process has access to an object (Ref 10). A third solution would be to store the access matrix by columns. Now each object is associated with a list of pairs (D,A[D,X]). A list of this type is often called an access control list or an authority list (Ref 11:45).

A combination of C-lists and access control lists have been implemented in the MULTICS Data Security System. This system gives MULTICS claim to the most secure large-scale general purpose system on the market today (Ref 12).

The access matrix model does not include mechanisms or rules corresponding to the requirements for military security. In systems based on the access matrix model, the protection of a file of information is the responsiblity of the file's owner. He can grant access to any user and any user granted read-access to the file can copy and distribute the information any way he pleases (Ref 1:257).

ADVANTAGES

Since the access matrix model specifies only that there are rules, subjects, objects and access attributes but not what the rules, subjects, objects or access attributes are in detail, the model has great flexibility and wide applicability (Ref 1:256).

Access attributes would best be represented as a binary word whose ith bit is 1 if and only if the ith access attribute is present (Ref 11:425).

The C-list implementation is inherently more efficient than the access control list implementation because a C-list must be stored in fast-access memory whenever a domain is active. However, an object may not be active often enough to warrant keeping its access control list in fast-access memory at all times (Ref 11:426). It is also important to note that the same techniques used to make virtual memory efficient can be generalized to make the C-list implementation efficient (Ref 11:426).

Finally, the access matrix model has served as the

basis for a number of other models and development efforts in the data security arena.

UCLA DATA SECURE UNIX MODEL

The UCLA Data Secure Unix Model is a demand paged, kernel structured operating system based on the concept of an access matrix. The model is an attempt to demonstrate that verifiable data security with general purpose functionality is attainable in todays medium scale computing systems. More specifically, the UCLA system has the characteristic that data security, the assurance that data can not be directly read or modified without special permission, is enforced via a limited amount of kernel software (Ref 14:355). Tremendous efforts have been expanded to demonstrate that the security properties of the kernel software have been correctly implemented. In addition, the system was designed so that confinement could be demonstrated by audit of some additional, isolated code.

To achieve the above goals, a number of design and implementation principles were integrated into a single system. These include a tightly constrained base kernel, a second-level policy kernel, an operating system interface, implementation in the high-level language PASCAL and application of formal, semiautomated program verification methods to the source code of both kernels.

The UCLA Unix architecture contains a number of modules whose relation to one another is specified in

Figure 2.

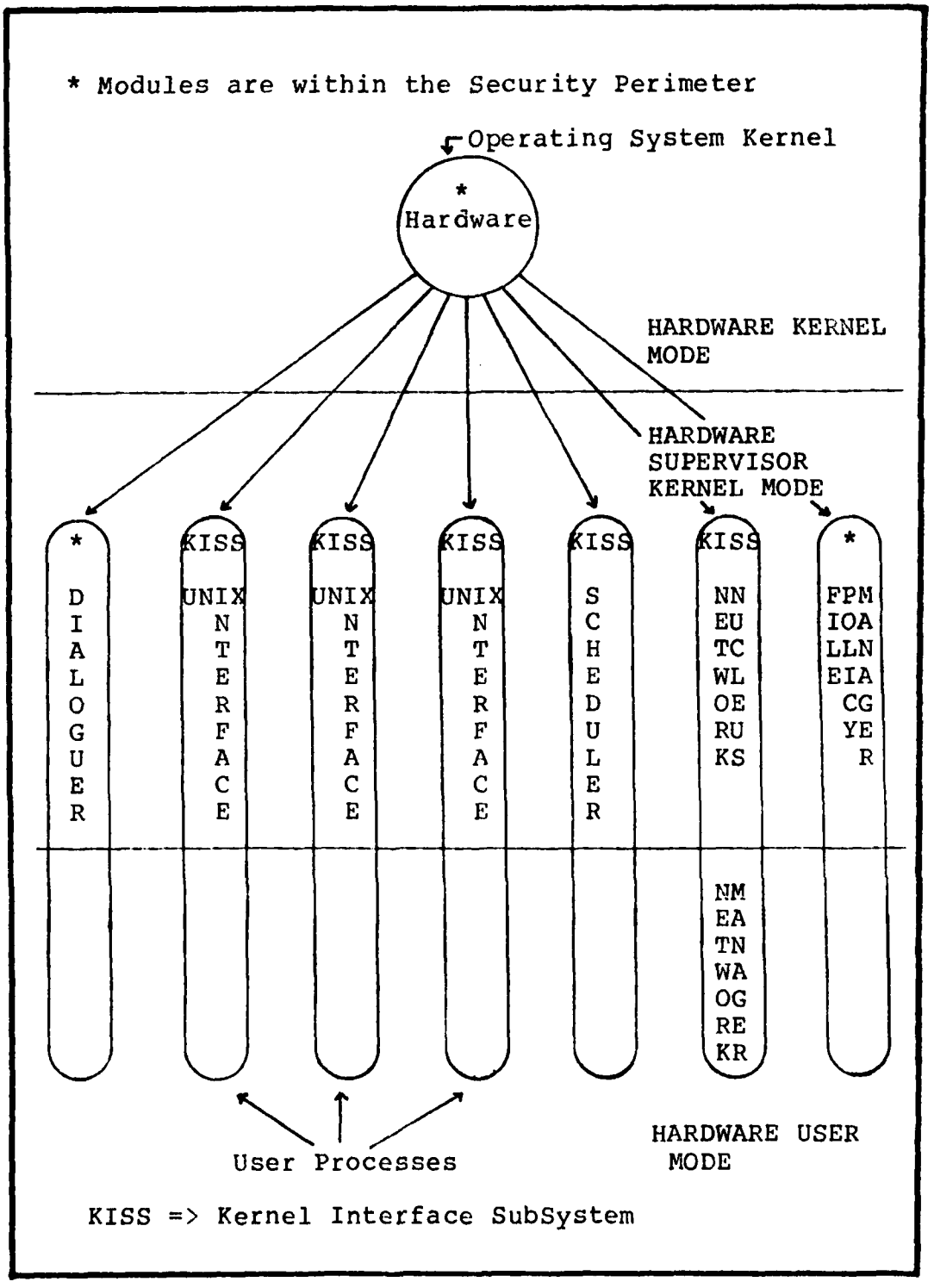


Figure 2 UCLA Secure Unix Architecture (Ref 14:356)

The kernel is to be thought of as an operating system nucleus which depends on the semantics of the hardware, thereby providing the base for the entire system. The kernel supports processes, capabilities, pages and devices via kernel calls. Examples of kernel calls to manipulate these objects are: initialize-process, invoke-process, grant-capability, swap-in-page, swap-out-page, memory-management-interrupt, send-interrupt, set-interrupt, hardware-interrupt and start-io. Part of supporting these objects involves enforcing an access policy, thereby protecting the objects.

One process implemented by the kernel contains two address spaces. An operating system interface package called the Kernel Interface SubSystem (KISS) resides in one address space. The other contains the application processes to be run. When an application process makes an operating system call, control passes to KISS which interprets the call. If necessary, the KISS issues kernel calls or uses kernel facilities to send messages to other processes to accomplish the needed action. All such calls or messages are controlled by the kernel. Each process is a separate protection domain. The access rights of the domain are represented by a capabilities list. Thus, a C-list for each process is maintained by the kernel (Ref 14:356).

A second kernel called the policy manager is

responsible for determining the protection policy. It is capable of altering the data upon which protection decisions are made and informs the kernel of the security policy, the rules which state what any user is entitled to access, by using the grant-capablity kernel call.

The two-level kernel design splits the task of proving that the system is secure into two subtasks: a) verifying that the kernel enforces the security controls set by the policy manager and maintains the integrity of the object types it supports; and b) verifying that the policy manager sets the kernel controls to implement the desired policy properly (Ref 15:119). As a result the overall system security depends on the correct operation of the two-level kernel design.

Both the kernel and the policy manager are implemented as sequential software but in different ways. This allows the specification and verification tasks to avoid the difficulties of parallelism.

Two other important modules in the Secure Unix architecture are the dialoguer and the scheduler. The dialoguer initially owns all terminals (has capabilities for all of them) and is responsible for user authentication. It tells the policy manager what user is to be associated with a given process. The scheduler, which handles memory and CPU allocation, is not relevant to data security because while it makes decisions about what process to execute and what pages to locate in core, all

the actual sensitive operations are done via kernel calls. Since the UCLA Unix is a demand paged system, when a process page faults, the scheduler is informed by the kernel so that an appropriate swap call may be issued at some later time by the scheduler.

Under normal operating conditions a user first logs into the dialoguer. That process sends a message to the policy manager, who initializes a process for the user and moves the user terminal to the new process by issuing appropriate capabilities. Process initialization as well as normal computation takes place within the domain of the given process. Additional resource requirements or file activity is done through messages to the policy manager. Process switching occurs whenever a given process invokes the scheduler process or when an appropriate clock interrupt forces such an invoke. The scheduler can then run whatever process it wishes. Page faults also force an invoke of the scheduler, so it can initiate appropriate page swapping (Ref 14:356).

The proof that the kernel is secure has two parts: a) developing four levels of specifications ranging from PASCAL code to the top-level security criterion; and b) verifying that the specifications at these different levels of abstraction are consistent with one another (Ref 15:119). Figure 3 outlines the four levels of specifications and the three consistency proofs.

The top-level specifications are formulated as a

finite state machine with the effect of each kernel call reflected in the transition function. The transition

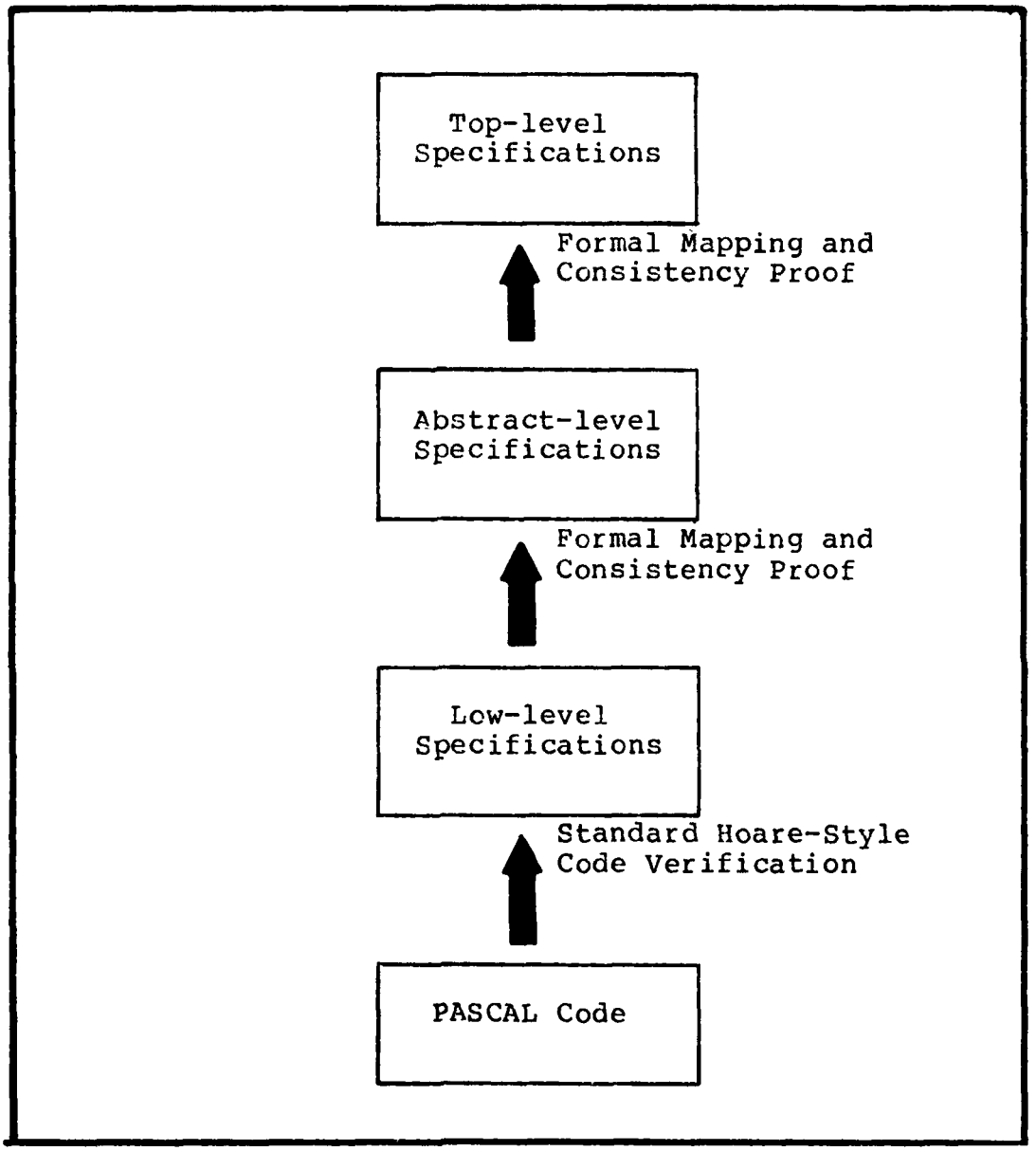


Figure 3: Specification Levels and Consistency Proofs (Ref 15:120)

function expresses the effect of a single operation which in turn corresponds to a number of operations at the abstract level.

A state of the machine consists of the following uniquely named, non-overlapping components:

a) Process objects, which at this level have unstructured values;

b) Protection-data objects, with values being sets of capabilities;

c) General objects, comprising both pages and devices; and

d) A current-process-name object, whose value is the name of the process currently running.

The top-level security criterion is as follows: a component of the state is actually modified or referenced only if the protection-data for the process named in the current-process-name object allows such access to the component. Now the only concern is whether a component is to be modified or referenced. If either, then that access would be allowed (Ref 15:120). In summary, this level of specification formalizes the intuitive notion of data security for the kernel.

The abstract-level contains operations corresponding to uninterruptable kernel calls and more specific objects like pages, processes, and devices. It, too, is given as a finite state machine. The low-level specifications, which use data structures from the implementation, are still higher than the code because detail is omitted and some abstracting of data structures is employed. Finally, PASCAL code which is the lowest level of kernel implementation. This level takes the actual low-level

specification data structures and creates detailed code for machine language conversion implementations.

All of the specifications at all of the levels, with the exception of the code, are written in predicate calculus. However, since the data structures are different at different abstraction levels, a mapping function from each specification level to the next higher level is used to construct higher-level data structures out of lower-level ones. Such a mapping can be used to show that two levels of specification have the same meaning. As a result several levels of specification and mappings can be used to verify that the top-level specifications are being correctly implemented by the PASCAL code for each kernel call (Ref 15:125).

The verification procedures of Alphard have been adapted as a basis for doing the mappings and consistency proofs. The Alphard methodology provides a formal framework to prove that abstract specifications of an abstract type are consistent with the code that implements that abstract type (Ref 16:260).

DISADVANTAGES

The model does not permit central processor asynchrony, for example multiple CPU's or interruptable kernel code, and so in its present form is limited in the types of systems for which it is useful (Ref 17).

The kernel and verification goals impose significant

constraints on the size, complexity, and general architecture of the system.

It was possible to verify the realistic software system that performs a reasonably complex task with suitable performance. However, highly skilled researchers were needed in the system's development, specification, and verification. A masters and a doctorial thesis were produced and four or five man years were spent in the overall verification project (Ref 15:128).

Specification methods need to exhibit considerable flexibility. For example, the intrinsically interrelated data structures are not compatible with the concept of abstract data types. To overcome this problem the low-level specifications need to be organized by data structure. This will permit mechanical transformation into the more conventional call-by-call organization needed by the verification systems.

The performance of the completed security system is poor, an order of magnitude slower than the standard Unix. However, the causes are either not related to the verification or could be easily remedied through modest changes in the hardware base. Two principal problems, language difficulties and hardware/software mismatches were identified. As an example of the programming language problem, the PASCAL compiler did not automatically generate machine code for array subscript bound checks, so many inefficient source code checks were implemented in the

procedure calls. Also, in-line procedure and function calls were not supported, which caused the majority of system execution time to be spent in procedure linkage overhead. Second and more serious was the domain switch overhead. A kernel-based architecture achieves minimization of security-related code at the cost of more frequent domain changes. The overhead is of little consequence if the domain switches are relatively cheap. But this was not the case with the UCLA Data Secure Unix system (Ref 15:128).

Finally, it was noted that the actual kernel implementation did not permit the creation or destruction of objects. The lack of creation and destruction merely corresponded to the static allocation of object-descriptor tables in the kernel. To compensate for this, the system architecture was carefully designed to permit large numbers of objects to exist efficiently. Also, the reuse of object-bodies, such as processes, was done at a higher level within the system (Ref 17).

ADVANTAGES

The UCLA Data Secure Unix is one of the first real operating systems designed and implemented with the concept of security in mind. All other previous attempts in trying to make operating systems secure merely found and fixed flaws as they occurred. Since most of these efforts failed, it became very apparent that piecemeal alterations would never solve security problems (Ref 18:23).

The two-level kernel design of splitting the security-relevant code provides several advantages. First, splitting a verification task in general decreases the total effort involved. Second, different policies can easily be implemented. Third, verifying that the implemented policy reflects a given abstract policy is not encumbered by the relatively complex task of enforcing that policy. Finally, and most important, the abstract specifications for the enforcement portion can be expressed straight forwardly and subsequently proven, independent of policy issues (Ref 19). As a result, the kernel type structure is exceedingly simple, yet robust enough for fairly general operating system activity. Also, the entire kernel is small enough to be locked down in main memory, blocking circular dependencies (Ref 14:359).

Page faults involving kernel-primitive instructions appear to user processes just as page faults involving hardware implemented instructions, thus completely transparent to the user (Ref 14:359).

The UCLA kernel was developed to be a candidate for firmware implementation. This was possible because each call behaves like a separate instruction, with no interruptions required while in execution. Also, there is no requirement to issue I/O calls for which the results affect the instruction's behavior, since I/O is typically slow relative to microprogram cycle speeds (Ref 14:359).

The array subscript checking problem, previously

mentioned as a disadvantage, could be easily remedied in one of the following four ways: first, by the hardware; second, by runtime software generated by the compiler; third, by runtime software explicitly inserted by the programmer; or fourth, by verifying that the subscripts do not get out of range (Ref 14:364).

An ARPANET connection is provided by the UCLA Data Secure Unix. Access to the ARPANET must be controlled and support for initial network connection activities is required. A capability based encryption is used to protect each connection individually (Ref 14:361).

Finally, the UCLA Data Secure Unix comprises the first verifiably secure, full function multiprogramming, uni-processor operating system running on a DEC PDP-11/45 computer, with application code interface essentially identical to the standard Unix (Ref 15:119).

TAKE-GRANT MODEL

The Take-Grant Model is a capability-based (C-list) protection system that uses graph theory to model a class of access control protection mechanisms in which each entity is protected independently of all others (Ref 21:238). Although couched in terms of graph theory, the Take-Grant model is fundamentally an access matrix model (Ref 1:258).

The Take-Grant model's capability-based protection system is described as a mechanism which limits the access of information to those entities, for example users, that have the "right" to access it (Ref 11:45). The "rights" are usually represented as tokens and the system keeps record of which entities have which rights. Within the capability-based protection system there are two distinct ways in which a user that does not have the right to read a file can acquire the information. The first is called de jure acquisition which means that the authority or right to read the information is transferred to the user. For example, some user that has the read right can grant it to another user who invokes the right and reads the information. The second is called de facto acquisition which means the user acquires the information in the file without necessarily acquiring the direct authority to access the file. For example, some user may be given a copy of the file from another user who does have the right to read it. The copy could then be passed via a mailbox

file common to both users (Ref 22:45).

These acquisitions are illustrated diagrammatically in Figures 4 and 5. The letters represent rights where r, w, and g abbreviate to read, write, and grant, respectively.

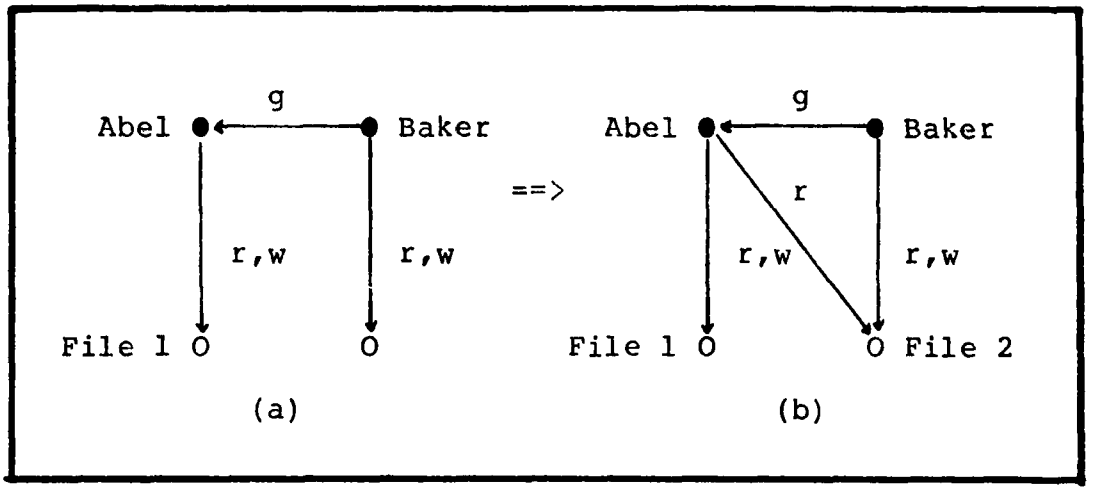


Figure 4: A de jure acquisition (Ref 22:46).

In Figure 4, the protection state is changed when Baker performs a de jure transfer by granting Abel the read right to File 2.

Figure 5 illustrates that there is a potential for de facto transfer of File 3 to Baker since Charlie could copy the contents of File 3 into the Mailbox. The dashed line emphasizes that no change in the protection state has taken place (Ref 22:45).

Although de jure an de facto acquisitions accomplish the same objective, of moving information, they are quite different. For instance, de jure acquisitions imply de facto acquisitions but not vice versa. Also, de jure

acquisitions always give the right to obtain an up-to-date version of the information whereas de facto acquisitions

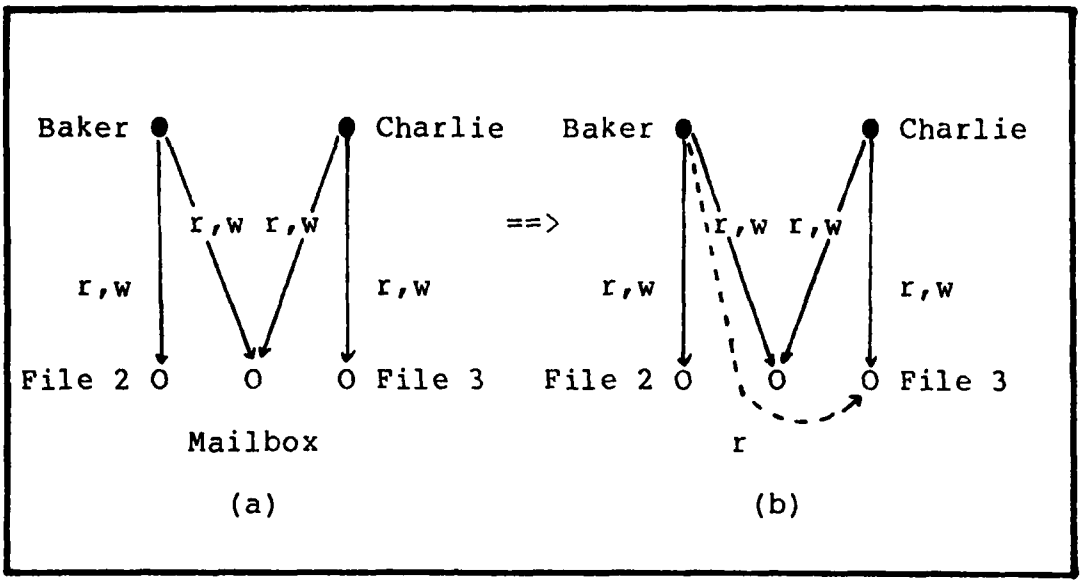


Figure 5: A potential de facto acquisition (Ref 22:46).

will not reflect any changes made to the original file since the copy was created. Futhermore, de facto acquisitions rely on agents to transmit a complete and accurate copy of the file.

The state of a Take-Grant protection system is a finite, directed, edge labeled graph called a protection graph. There are two types of vertices in the protection graph, subjects and objects. Subjects (denoted by ●) represent active entities such as users, and objects (denoted by O) represent passive entities such as files. A third vertex (denoted by ●) representing either subjects or objects may also exist.

The edges between the vertices are labeled with elements from a finite set R corresponding to rights. For

instance, assume R contains the letters r, w, t, and g, mnemonics for read, write, take, and grant, respectively. Normally, edges are either explicit or implicit. Explicit edges represent authority that is formally recorded in the protection system. They are associated with de jure acquisitions because they cause an authority change in the protection graph. Implicit edges also represent authority that is formally recorded in the protection system. However, they are associated with de facto acquisitions because they do not cause an authority change in the protection graph.

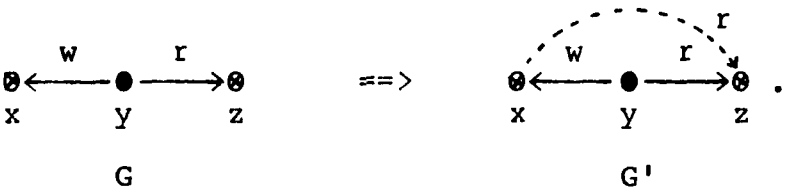
There are four rewriting rules for de facto acquisitions. They are post, pass, spy, and find.

POST: Transfers abstract parts of the operation via a mailbox. For example, let x, y and z be distinct vertices in a protection graph G such that x and z are subjects. Let there be an edge from x to y labeled A, where r is an element of A, and an edge from z to y labeled B, where w is an element of B. Then post defines a new graph G' with an implicit edge from x to z labeled r. Graphically,

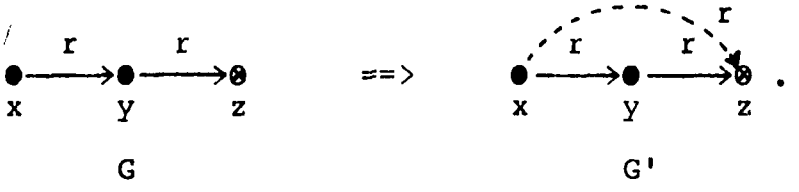
G G'

PASS: Acts as a conduit through which information

travels. For example, let x, y and z be distinct vertices in a protection graph G such that y is a subject. Let there be an edge from y to x labeled A, where w is an element of A, and an edge from y to z labeled B, where r is an element of B. Then pass defines a new graph G' with an implicit edge from x to z labeled r. Graphically,



SPY: Abstracts information by watching. For example, let x, y and z be distinct vertices in a protection graph G such that x and y are subjects. Let there be an edge from x to y labeled A, where r is an element of A, and an edge from y to z labeled B, where r is an element of B. Then the spy rule defines a new graph G' with an implicit edge from x to z labeled r. Graphically,



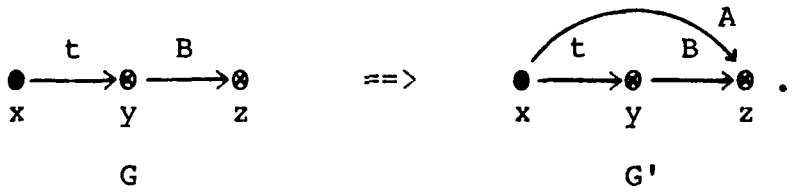
FIND: Abstracts information and then deposits it somewhere else. For example, let x, y and z be distinct vertices in protection graph G such that y and z are subjects. Let there be an edge from y to x labeled A, where w is an element of A, and an edge from z to y labeled

B, where w is an element of B. Then find defines a new graph G' with an implicit edge from x to z labeled r. Graphically,

(Ref 22:47).

There are also, four rewriting rules for de jure acquisitions. They are take, grant, create, and remove.

TAKE: Causes a user to acquire a new right over some system entity. For example, let x, y and z be three distinct vertices in a protection graph G such that x is a subject. Let there be an explicit edge from x to y labeled C, where t is an element of C, an explicit edge from y to z labeled B and A is a proper subset of B. Then the take rule defines a new graph G' by adding an explicit edge to the protection graph from x to z labeled A. Graphically,

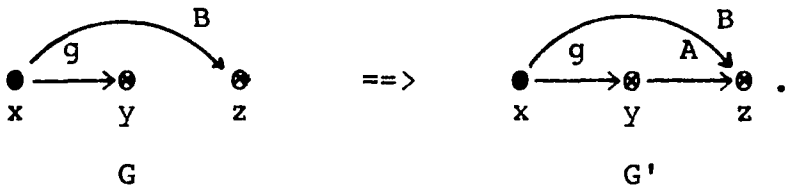


.

Read: x takes (A to z) from y.

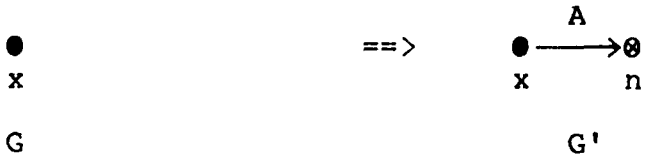
GRANT: Gives some right away. For example, let x, y and z be three distinct vertices in a protection graph G such that x is a subject. Let there be an explicit edge from x to y labeled C, where g is an element of C, an

explicit edge from x to z labeled B, and A is a proper subset of B. The grant rule defines a new graph G' by adding an explicit edge from y to z labeled A. Graphically,



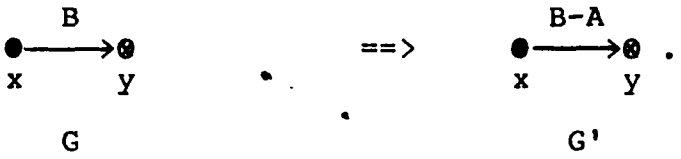
Read: x grants (A to z) to y.

CREATE: Enables new processes and files to have their protection configuration added to the system structure. For example, let x be any subject vertex in a protection graph G and let A be a subset of R. Create defines a new graph G' by adding a new vertex n to the graph and an explicit edge from x to n labeled A. Graphically,



Read: x creates (A to) new subject or object n.

REMOVE: Eliminates rights. For example, let x and y be any distinct vertices in a protection graph G such that x is a subject. Let there be an explicit edge from x to y labeled B, and let A be any subset of rights. Then remove defines a new graph G' by deleting the A labels from B. If B becomes empty, the edge itself is deleted. Graphically,



Read: x removes (A to) y (Ref 22:50).

There are also several important characteristics concerning the de jure acquisition rewriting rules.

a) Take and grant do not create any new rights -- they merely disseminate existing rights.

b) Once all rights to a subject or object have been removed they can never be restored.

c) Rights once granted away can never be recovered. That is, they can be distributed by the grantee without consultation with the grantor (Ref 23:144).

NOTE: The de jure acquisition rules can only match explicit edges while the de facto rules can match explicit or implicit edges.

In summary, the protection system defined by the Take-Grant model is a logically separate entity from the operating system supervisor. As a result, the independence of the protection system allows the user to query the system himself for an audit to verify that certain protection conditions hold. The protection graph is a description of the currently existing protection relationships. Thus, the protection relationships among system entities can be changed by the de jure or de facto acquisition rewriting rules. The subjects are generally thought to be user processes or components that are active

from a protection point of view, while the objects are thought of as files or processes known to be secure. Since only subjects can apply the rules, a subject applying a rule may be requesting a modification of the protection state.

DISADVANTAGES

Generally, security policies are formulated as predicates relating the subjects and passive objects of a protection state. In contrast, most protection mechanisms, and the Take-Grant system, are phrased in a procedural, not a predicate, form. Though procedural definitions make individual system state transitions easy to understand and to implement, they combine to form a system that exhibits complex behavior (Ref 21:242).

The de facto acquisition rules pose a minor problem in the Take-Grant model. For example, if de facto transfer occurs, which involves the movement of information rather than authority, it is impossible to tell by observing the protection graph whether or not the de facto transfer actually took place. The only inference to be made at this point is, given a particular arrangement of authority relations a de facto transfer could occur.

Finally, the major objection to using Take-Grant models is that they are too weak to provide useful inferences about protection systems. Most claims emphasize that the result of applying a Take-Grant model to a "real"

system will be a fully connected graph in which all the subjects can gain access to all the objects. Consequently, the model appears limited in respect to its ability to handle controlled sharing of information (Ref 1:260).

ADVANTAGES

Earlier in the chapter two types of protection mechanisms were defined that could be used to control access to objects. One was called the access control list mechanism and the other was called the capability-based mechanism. The only difference between the two is where the data that records permissable accesses is stored. Therefore, either may be used in the Take-Grant model, although the capability-based mechanism proves to be more efficient.

Previously, most of the available formal security models implementing the capability-based protection systems focused only on the de jure acquisition rewriting rules. However, the Take-Grant model is the forerunner in successfully implementing both the de jure and de facto rewriting rules. An additional benefit is that the de facto rules can provide either one-way or two-way communication between subjects, previously not possible (Ref 22:53).

Most research literature defines take, grant, create, and remove as the key de jure operations. However, other rules may be added depending on the properties of the system being modeled (Ref 1:259). Thus, the Take-Grant

model offers flexibility.

The Take-Grant model does not include security levels. However, the levels could be easily added by labeling subjects and objects and by restricting the graph rewriting rules according to some set of security relations.

Finally, one of the most important questions asked of the Take-Grant model was: Given an initial protection graph and the set of rules, is it possible for a subject A to gain a particular access right to an object B? Previous research showed this problem to be undecidable for an arbitrary set of rules and an initial graph but decidable for a specific set of rules (Ref 24:468). The rules of the Take-Grant model demonstrated that the answer to the above question could be computed in a time directly proportional to the size of the graph (linear time) (Ref 25:37).

HIGH-WATER MARK MODEL

The high-water mark model was one of the earliest formal security models implemented. This model is mainly concerned with the security levels of the objects which a domain touches over time while controlling direct access to objects (Ref 1:273).

The ADEPT-50 time-sharing system was an attempt to implement software controls for classified information. Although the system was never certified by DOD for operation as a multilevel secure system, its controls were

based on the high-water mark model (Ref 1:255).

The ADEPT-50 security model defines security control of user, terminal, job and file security objects in terms of equations of access based upon security profiles. Each object is described by a security profile that consists of an ordered triple of security properties: Authority; Category; and Franchise (Ref 1:255). Authority is a set of hierarchically ordered security sensitivity levels which correspond to the Top Secret, Secret, Confidential and Unclassified levels of government and military security. Category is a discrete set of specific compartments which are used to restrict access by project and area. Franchise consists of a set of user designations which correspond to access sanctioned on the basis of a need-to-know. The model also defines an ordering on the triplets mentioned above.

"History functions" are defined for the authority and category properties of an object. These functions record the highest authority assigned to the object and the union of all categories assigned to the object since its creation. These are referred to as the high-water mark of the object, from which the model assumes its name (Ref 1:255).

The values of the current Authority, Category and Franchise properties and the history functions are used to control the properties assigned to new objects (for example, newly created files) and to determine whether

requested operations will be allowed.

In order to access the ADEPT-50 system the user must present a user ID and a password. The system then checks a list that is stored at system start time to see whether this ID is known to the system and is in the franchise set for this terminal. It also checks the password at this time. If logon is successful the job is assigned the minimum of the authorities assigned to the user and the terminal. The job is also assigned a category set corresponding to the intersection of the user and terminal category sets. Permission for this job to access a file is granted if and only if the level of the job is at least that of the file. Granting access to a file causes the history functions to be updated according to the authority and category set for that file. New files that are created by this job are assigned an authority based on the highest file accessed by this job since logon; the category is set to the union of the category sets of all files accessed since logon; and the franchise is set to that of the job.

DISADVANTAGES

The possibility of a Trojan horse existing within the system would allow copying of classified information to an already existing file that is unclassified. This copying of information would be possible because the rules of the model allow authorized "downward" flows of information.

The controls governing classification of new files can lead to overclassification of data since the high-water

mark can never decrease during a given run. This could result in downgrading of classified data, which would make errors or intentional violations in downgrading harder to detect (Refs 1 and 13).

ADVANTAGES

The high-water mark model is one of the few models that have actually been implemented on an operational computer system, namely the IBM 360/50.

The authority, category and franchise properties of the model are adequate when describing the static structure of military security. The restriction that a user can only have access to a file at or below his level in the system ensures he cannot directly read information contained in a file classified above his clearance level.

Finally, ADEPT can be looked at as a multiuser, multiprogram, multifile security system.

BELL-LAPADULA (ORIGINAL) MODEL

Since the early 70's the Electronic Systems Division (ESD) of the United States Air Force, the MITRE Corporation, and the Case Western Reserve University have been involved in various projects relating to secure computer systems design and operation. One of the continuing efforts, started in 1972 at MITRE, has been secure computer systems modeling. The effort initially produced a mathematical framework and a model by D. Elliott Bell and Leonard J. LaPadula called the Bell-LaPadula model

(Ref 27:5). This model used a finite-state machine to formalize an access matrix model for incorporating military security policies.

The model is viewed as having four major facets: elements, invariant system characteristics, general mechanisms, and specific solutions. Each facet will be discussed in detail.

ELEMENTS

The model represents abstractly the elements of computer systems necessary to study security issues. The active entities in the system are called subjects (for example, users, programs, and processes) and the passive entities are called objects (for example, I/O devices, files, and source code). Some entities can be both subjects and objects. The security problem that exists is to control access of subjects to objects, where this control is based on some security policy. For example, a specific security policy might be to control the unauthorized observation of information.

Security threats can be of a direct or an indirect nature. If the direct observation of information without authorization takes place a direct security threat has occurred causing a breach of security. On the other hand, if classified information is allowed to be manipulated or moved in such a way that unauthorized observation is possible then an indirect security threat has occurred.

Both direct and indirect threats can be controlled by forcing an observation to satisfy static requirements for appropriateness, as well as requiring explicit permission.

As in computer systems, access in the Bell-LaPadula model can assume different modes. These modes of access are distinguished by the varying effects that different accesses have on objects. Usually, the two effects that accesses have on an object are the extraction of information (observing the object) and the insertion of information (modifying the object). Thus, there are generally four types of access defined:

1) No observation and no modification;
2) Observation but no modification;
3) Modification but no observation; and
4) Both observation and modification.

In addition, an access mode for each of the above access types is included in the model:

1) Execute access (e) => Neither observation or modification;
2) Read access (r) => Observation but no modification;
3) Append access (a) => Modification but no observation; and
4) Write access (w) => Both observation and modification (Ref 27:11).

A system state in the Bell-LaPadula model can be expressed as a set of four values, each referred to as a component. The first component of a system state is the current access set, denoted by b. A current access by a subject to an object is represented by a triple: (subject, object, access-mode). The triple means that "subject"

currently has "access-mode" access to "object" in the state (Ref 27:11). Thus, the current access set b is a set of such triples representing all current accesses.

The next state component involves access permission. It allows discretionary (dynamic) control of object access. That is, a subject may access an object only if permission to do so has been explicitly given by a subject authorized to give this permission. Access permission is modeled as a matrix M, where component Mij records the subset of access modes in which subject Si is permitted to access object Oj.

The third component of a system state is a level function, which represents the embodiment of security classification in the model. This function (f) assigns to each subject and each object a security level. The level function F is a triple of (fs, fo, fc) where

> fs equals the maximum security levels of all subjects;
> fo equals the security levels of all objects; and
> fc equals the current security levels of all subjects (Ref 28:11).

A security level is represented as a pair: (classification, set of categories). In a military or governmental environment, classification is one of the ordered set {unclassified, confidential, secret, top secret} and a set of categories is a set of formalized need-to-know compartments. The purpose of the security level is to embody a security policy by allowing the assignment of levels to restrict observation of objects in a non-discretionary (static) way. This can be accomplished by

requiring that a subject may observe an object only if the subject's security level dominates the object's security level.

The last component of a system state concerns the structure imposed on the objects. The access control of an object is built into the structure by a hierarchial organization, H, where access to any object becomes dependent on its parent object.

Finally, the representation of a system state is a four-tuple of the form: (current access set, access permission matrix, level function, hierarchy). Thus, the model notation for a state is (b,M,f,H). Subsequently, a relation W must be defined, which specifies which characteristics (collectively known as security) are to be maintained by the system. Inputs to the system are requests and outputs are decisions. Now the system contains all sequences of (request, decision, state) triples with some initial state which satisfies relation W on successive states (Ref 28:12).

INVARIANT SYSTEM CHARACTERISTICS

There exist four invariant system characteristics which are very important to the proper functioning of the Bell-LaPadula model.

The first characteristic, the simple security property (ss-property), is needed to ensure a state is secure. The ss-property is satisfied if every "observe"

access triple (subject, object, access) in the current access set b has the property that the maximum level of the subject dominates the level of the object (Ref 28:13). In sum, no subject has read access to any object that has a classification greater than the clearance of the subject.

The *-property (pronounced "star-property") is the second characteristic considered. It too, is needed to ensure a state is secure. The *-property tries to prevent any potential, indirect security threats from compromising information. These indirect threats can be avoided by conforming to the *-property, which is satisfied if: in any state, if a subject has simultaneous observe access to object-1 and modify access to object-2, then the security level of object-1 is dominated by the security level of object-2 (Ref 28:14). More specifically, the *-property ensures that:

> No subject has append access to an object whose security level is not at least the current security level of the subject;
>
> No subject has read-write access to an object whose security is not equal to the current security level of the subject; and
>
> No subject has read access to an object whose security level is not at most the current security level of the subject (Ref 1:261).

In addition, two important comments need to be made about the *-property. First, it does not apply to trusted subjects (a trusted subject is one guaranteed not to consummate a security-breaching information transfer even

if it is possible). Second, both the ss-property and the *-property must be enforced to ensure a state is secure.

The third characteristic is the tranquility principle which states that the security levels of active objects will not be changed during normal operation (Ref 29:244). The tranquility principle disallows the classification of objects to vary according to the accesses performed on them. This is essential so that objects will not be downgraded. Additionally, upgrading data can result in the overclassification of information, reducing its usability.

The discretionary security property (ds-property) is the last of the invariant system characteristics. It allows an individual to extend to another individual access to objects based on his own discretion, but constrained by a non-discretionary security policy (requires the existence of the ss-property, the *-property, and the tranquility principle). Specifically, the ds-property requires that: if (subject-i, object-j, access-x) is in b, then access-x is recorded in the (subject-i, object-j), thus x is in Mij (Ref 28:16).

GENERAL MECHANISMS

The first general mechanism in the Bell-LaPadula model is the basic security theorem which states that security can be guaranteed systematically when each modification to the current state does not itself cause a breach of security (Ref 28:16). Thus, the preservation of security from one state to the next guarantees total system

security.

The second general mechanism within the Bell-LaPadula model is a direct consequence of the basic security theorem. Since the systematic problems of security can be dealt with one state transition at a time, a general framework for isolating single transitions is needed. This framework relies on the "rule", a function for specifying a decision (an output) and a next state for every state and every request (an input) (Ref 27:21):

$$(\text{request}, \text{current-state}) \overset{\text{Rule}}{\Longrightarrow} (\text{decision}, \text{next state}).$$

Now each class of requests can be analyzed separately by a rule designed to handle that particular class. For clarity, no two rules will be allowed to specify non-trivial changes for a given (request, current-state) pair.

Finally, the last general mechanism centers on the relation of rule properties to system properties. This can be shown by a requisite demonstration that a rule which preserves security can, in most cases, be reduced to the direct consideration of a small number of state modifications involving a given state transition.

SPECIFIC SOLUTIONS

The rules presented below represent a specific solution to the requirement for a secure computer system. This particular solution is not unique but has been specifically tailored for use with a Multics-based

information system design. As a result, the solution has to satisfy two requirements: the provision of generally useful functions; and appropriate accommodations to the effects of the Multics design on an implementation of the Bell- LaPadula model (Ref 27:23).

A number of general functions can be suggested for any computer-based information system. In reference to the Bell-LaPadula model these functions can be grouped into four classes:

A. Functions to alter current access (the set b):
   1) To get access (add a triple "(subject,object, access)" to b), and
   2) To release access (remove a triple from b);

B. Functions to alter the security levels of subjects and objects:
   1) To change object level (change the value of fo (object) for some object), and
   2) To change current level (change the value of fc (subject) for some subject;

C. Functions to alter the current access permission structure (the matrix M):
   1) To give access permission (to add an attribute to some component of the access permission matrix M), and
   2) To rescind access permission (to delete an attribute from some component of M); and

D. Functions to alter the object structure (the hierarchy H):
   1) To create an object (to attach an object to the current tree structure as a leaf), and
   2) To delete a group of objects (to detach from the hierarchy an object and all other objects "beneath" it in the hierarchy (Ref 28:19).

These rules reflect several characteristics of the Multics operating system. The main Multics characteristic that affects the revised model is the hierarchical object

structure. In addition, the basic Multics mechanisms for access control rely heavily on this object structure. The second Multics characteristic involves the implemented counterpart of the access permission matrix M. This structure is called the Access Control List (ACL), with one ACL stored with every object. Each ACL is a list of processes (subjects) allowed to access the corresponding object, as well as the allowed modes of access. These ACL's are contained and manipulated in an objects parent object (for example, a directory object) (Ref 28:19). Therefore, control over an object is equivalent in Multics to write permission to the object's parent. Since object creation in Multics is the insertion of a new entry in the parent object, creation control is equivalent to append access to the object's parent.

The final Multics characteristic reflected in the Bell-LaPadula model is the way access to an object is performed. A user request to access an object causes the user's surrogate (process) to access every object in the hierarchy in the path from the root directory to the segment of interest. The rules of the model require that the security level of an object dominate that of its parent object (Ref 27:29).

In conclusion the Bell-LaPadula solution provides a particular specification for a secure computer system that supplies a full complement of information processing capabilities while matching the special requirements of the

Multics operating system environment.

DISADVANTAGES

The original Bell-LaPadula model considers only a flat set of objects. However, objects are considered to be atomic elements, each with a single classification and containing no distinguishable subelements. Consequently, hierarchies of objects may be incompatible.

The static representation the model provides for military security is restrictive. Although hierarchies of objects are provided, the Bell-LaPadula model does not lay out an appropriate set of axioms governing references to multilevel objects (Ref 1:262).

The dynamics of security, reclassification, sanitization, and downgrading can only be handled using the trusted process concept. Unfortunately, the model gives little guidance for determining which processes can be trusted (Ref 1:263).

Some of the rules in the model have been found to allow information to be transmitted improperly through control variables such as storage channels (Ref 1:263).

Programs that appear to be insecure in terms of the model have been proven to be secure. This is possible because the Bell-LaPadula model provides constraints that are sufficient, but not necessary.

Variables within the security kernel can act as information paths, in which information recorded in a

kernel variable as a result of a kernel call by a Process A might be visiable later to a kernel call by Process B. Now if B has a lower security level than A it is possible that a security violation has occurred (Ref 1:263).

The hierarchical tree-structured directory does not permit searching for all dominated objects, which is often desirable in a general data management system.

It is also important to note that in the Bell-LaPadula model an object can not be accessed until every directory object in the unique path to the object has been accessed. Consequently, additional processing time is required.

In summary, the principal disadvantages with the original Bell-LaPadula model are not in the things it allows but in the things it disallows. For instance, many operations will appear to be insecure to the model and disallowed when in fact they actually are secure and should be allowed.

ADVANTAGES

The Bell-LaPadula model has proven to be a significant advancement in computer security by defining military security concepts in a way applicable to computer systems.

The particular set of rules (specific solutions) developed and proved by Bell and LaPadula is not integral to the model and is not generally used. However any system based on an access matrix model will have similar rules

(Ref 1:262).

If additional restrictions are added to the concept of security the impact on the ss-property, the *-property, the tranquility principle, and the ds-property, is minimal because any additional restrictions can only reduce the set of reachable states (Ref 28:16).

Finally, the Bell-LaPadula model has served as the basis for several design, prototype, and implementation efforts. For instance, both the original and the revised Bell-LaPadula models or modifications thereof have been incorporated into the security enhancements of Multics for the Air Force Data Services Center, the MITRE brassboard kernel, the SIGMA message system used in the Military Message Experiment, the Kernelized Secure Operating System (KSOS) for the PDP-11, the Secure Communications Processor (SCOMP) for the Honeywell Level 6, and the Kernelized VM/370 (Ref 1:262).

BELL-LAPADULA (REVISED) MODEL

Since work initially began on secure computer systems, an application that has been of particular interest, in the computer industry, is the implementation of a Secure Database Management System (DBMS) (Ref 1:266). As part of the work sponsored by the Air Force to develop a secure version of MULTICS, the Bell-LaPadula (original) model was revised for relational database implementation on top of a Multics file system. Consequently, the Bell-LaPadula (revised) model evolved.

As a basis for the revised model, the original model was used in order to take advantage of: the work that had already been done; the recognized terminology; and the techniques that had already been proven. In addition, the revised model was to be as small and as simple as possible. As a result, this would enhance the clarity of the model elements and operations, and simplify demonstrating the correspondence between specification and the model.

Since the security mechanisms defined by the original Bell-LaPadula model are not sufficient to permit adequate control over modification access, an additional mechanism called integrity needed to be implemented. Integrity, being a static property of a dynamic system, would ensure that the initial "soundness" of the system be maintained throughout its activities (Ref 28:21). However, modification control was needed because modifications potentially deteriorate the initial soundness of a system.

Therefore, the major purpose of integrity was to embody an integrity policy by allowing the assignment of levels to appropriately restrict the modification of objects in a non-discretionary (static) way. Thus the integrity policy was quite similar to the security policy established in the original model.

Data can be protected from direct unauthorized modifications provided the system maintains a simple integrity property which ensures that a subject can modify an object only if the maximum integrity level of the subject dominates that of the object.

Data can be protected from indirect unathorized modifications provided the system maintains a *i-property (read star-integrity property) which ensures that if a subject has simultaneous modify access to object-1 and observe access to object-2, then the integrity level of object-1 is dominated by that of object-2.

Together, the simple integrity property and the *i-property cause the integrity levels of all the objects accessed by a given subject to be ordered (Ref 28:24).

The tranquility principle for integrity is defined as for security, and states that the integrity levels of active objects will not change during normal operation. Consequently, the integrity levels ensure that clearance matching and the formal need-to-know requirements are observed.

The discretionary integrity policy is already

embodied in the discretionary security property established by the original model.

Since the authorization of every data access has to be checked, the procedure to determine whether or not the security and integrity properties are satisfied has to be as simple and efficient as possible. As a result, the security and integrity mechanisms are consolidated to produce simple properties involving a minimum of entities. These mechanisms are not easily combined for a type of access, since different levels (for example, subject security level, object integrity levels for observe) are involved. Therefore, it is necessary to redefine the properties in terms of the current levels of the subjects. This is done because the current level can represent the current "limits" of the levels of objects a subject will be permitted to observe or modify (Ref 30:18). The consolidation process proceeds as follows:

A protection level is defined to be an integrity level appended to a security level. Symbolically the jth protection level is:

Pj = (Cj, Kj, C'j, K'j) where

Cj = a security clearance or classification;
Kj = a set of security categories;
C'j = an integrity clearance or classification; and
K'j = a set of integrity categories (Ref 28:28).

Analogous to the original Bell-LaPadula level function f, subjects and objects are mapped into protection

levels by the function (H), which is a triple (Hs, Hc, Ho), where:

Hs = protection level limits for all subjects;
Hc = current protection levels of the active subjects; and
Ho = protection levels of all objects (Ref 28:28).

The main reason for the existence of current security levels was to allow dramatic simplifications of the *-property checks. For instance, if a subject wishes to write (both observe and modify) at a particular level, the subject will choose a current security level equal to that level. This follows from the fact that the subject can modify only dominating objects and observe only dominated ones. Thus the dual nature of the integrity properties with respect to the security properties suggests that protection dominance can be defined as a dominating security level and a dominated integrity level (Ref 28:29). With this definition of protection dominance, protection properties analogous to the security properties can be defined by consolidating properties regarding a given access mode. These new properties are as follows:

SIMPLE-PROTECTION PROPERTY

A subject may observe an object only if the current protection level of the subject dominates the protection level of the object. This definition embodies both the modified simple security property and the modified *i-property;

*'-PROPERTY

A subject may modify an object only if the protection level of the object dominates the current protection level of the subject. Again, this definition embodies both the modified *-property and the modified simple integrity property;

TRANQUILITY PRINCIPLE FOR PROTECTION

This principle states that the protection levels of active objects will not change during normal operation. It is required to ensure that objects are not downgraded. Also, this principle removes the risk of a varying level constituting a communication path; and

DISCRETIONARY PROTECTION

The discretionary protection property is defined to be the same as the discretionary security property, namely that if subject-i has x-access to object-j, the x is recorded in the (subject-i, object-j) component of M (Mij).

The simple protection property and the *'-property will cause the protection levels of all objects accessed by a given subject to be ordered. However, trusted objects are not included in the revised model because they are considered to be proven programs which are completely within the security perimeter and therefore not subject to the rules of the revised model.

In the Bell-LaPadula original model the basic access

control mechanisms relied heavily on the hierarchical structure of the directory system (Ref 27:21). In the revised model the directory objects will serve the same purpose, but changes are desirable for the following reasons:

1. The path through the hierarchy must be specified in order to access an object. However, it will be impossible to access the object unless the subject has discretionary access to every directory object in the path; and

2. The new directory system (unlike the hierarchical one) does permit searching for all dominated objects which is very desirable in a general data management system (Ref 28:33).

Although the directory system is completely transparent to the system user, its components are modelled to address protection issues adequately in this important system mechanism.

If dealing with an object in a data base using the revised model, it is necessary to distinguish between the descriptive and the values aspects of the object. As a result, this will allow a data base to be self-descriptive and therefore system independent. Thus it is necessary to define a protected object, O, having three aspects which can be represented as follows:

O = (Mo, Do, Vo) where

Mo represents the column of an access permission matrix;
Do represents the descriptive aspect, including current status; and
Vo represents the value aspect (Ref 28:38).

In the original model these three aspects of a protected object are not defined.

An object in the original model can not be accessed until every directory object in the unique path to it has been accessed. In the revised model the accessing of an object involves only one directory. This is not of interest to the end-user, because the revised model's directory system is invisible to them. However, it is relevant to the "Total" system point of view (Ref 28:39). Consequently, a direct access function is defined to serve the following functions: to model object access; to model directory access; to enforce protection requirements on accesses; and to allow the directory system to be invisible to the end-user (Ref 28:39).

The original model does not protect execute access with the simple security property or the *-property. As a result, potential protection threats exist. In the revised model the execute access is subject to the simple protection property (simple security and *i-properties), the tranquility principle and discretionary control. Thus the execute access is considered equal to the read access. In addition, many of the potential protection threats disappear.

A, in the original model, is a set of access attributes or modes in which a subject may access an object in the system. Specifically, A = {e, r, w, a}. However,

under the revised model this set can be reduced as follows: execute (e) access can be removed since it is identical to read (observation) access; and write (w) access can be removed since it is a combination of observation and modification (Ref 27:7). Now the revised model's access mode is equal to:

A = {o, m} where

o = observe access, which involves the extraction or utilization of data from an object; and

m = modify access, which involves modifying the object without any observation of it. This is often referred to as append (Ref 28:44).

It is also important to note that a destructive modification (replace) will require both an observe and a modify access. Thus this set of two access modes is sufficient.

Previously, in the original model, a request was an input to the system which had an associated rule which specifies the output. Such requests are grouped according to the model elements involved in servicing the request (rules). Since the revised model has a different set of elements, the set of requests will also be different. Thus the revised model requests are grouped according to type of access to the model elements which contain the information relevant to a data management system. Consequently, there are eight classes of requests when they are grouped by access on model elements. Each access request class consists of a set of requests which are

either an observe (O) or a modify (M) to one of the following: a directory (L); a permission matrix (M); a descriptor (D); or a value set (V). The eight classes and their rules are found in Table I.

Table I. Classes of Requests (Ref 28:46).

CLASS	ACCESS MODES	MODEL ELEMENTS
Q_L^O	Observe	Directory
Q_L^M	Modify	Directory
Q_M^O	Observe	Permission Matrix
Q_M^M	Modify	Permission Matrix
Q_D^O	Observe	Object Descriptor
Q_D^M	Modify	Object Descriptor
Q_V^O	Observe	Object Value Set
Q_V^M	Modify	Object Value Set

The original model was very detailed regarding the secure access of data objects and less detailed about the

nature of subjects. However, it is desirable that an active subject not change its protection level during normal operation. If it could then the success or outcome of a given program (active subject) could depend on whatever peculiar activities the process was previously involved in. This would make results less predictable. Additionally, varying protection levels could result in communication paths. Therefore, it is necessary that the tranquility principle be extended to apply to subjects as well as objects.

Since the revised model was to represent data management systems, data structures and operations had to be embodied in the model. Consequently, the relational approach to data management was chosen for the following reasons:

1. To provide a high degree of data independence. Then access paths can be established (independently of programs) to involve the data accessed most frequently:

2. To provide a simple and consistent data model, and make it broadly usable;

3. To introduce, through relational algebra, a theoretical foundation (albeit modest) into data base management; and

4. To raise application programming to a new level, where data (relations) are treated as operands instead of being processed element by element, in ad hoc ways (Ref 31:4).

The decision as to how a relational data base should be mapped into the set of protection levels was an

important issue. In the revised model, Bell and LaPadula arrived at a decision by studying the restrictions resulting from alternate mapping assignments. Two assignment possibilities (whole data bases and individual data values) are quickly rejected because of unacceptable disadvantages. Assigning protection to a whole data base (collection of relations) was too coarse an assignment and would impede data sharing. On the other hand, assigning protection to attributes would introduce an unacceptable amount of overhead in any proposed implementation (Ref 28:57). Therefore, it was necessary to choose a data structure between a data element and a data base. As a result, three different areas (tuple assignment, domain assignment, and relation assignment) for assigning protection levels were studied. The results are as follows:

TUPLE ASSIGNMENT

If protection levels are assigned to the tuples of a relation, there are restrictions on certain tuple activities. For instance, a tuple cannot change its protection level when such a change would constitute a communication path. A second restriction occurs in defining a directory system to support the assignment of protection to individual tuples. Now each relation might need several aliases, each having a different protection level. Consequently, the management of such a heterogenous directory/relation schema is very difficult. In addition,

the possibility exists that a high-level tuple must contain exactly the same data as a low-level tuple. But this situation cannot be accommodated in the revised model;

DOMAIN ASSIGNMENT

If protection levels are assigned to the domains of a relation, restrictions on activities arise. For instance, for a relation with domains of varying levels, for any observable tuple, only those domains with a protection level dominating the subject's are modifiable (*'-property). This can cause problems with tuple oriented reading or updating. For example, it may be necessary to insert a tuple into a relation a domain value at a time (when each has a different level), signing on a different level for each domain value. Consequently, this technique can be very inconvenient (Ref 28:59). If the prime key of a tuple is at a higher protection level than the other domains, the key will be unobservable to some subjects and the tuple will be invalid. Therefore, the formation of relations will be restricted to those in which all domain levels dominate that of the prime key; and

RELATION ASSIGNMENT

Assigning protection levels to relations is the best choice since it causes no restrictions on either tuple oriented or domain oriented activities. In addition, this assignment is consistent with the use of relations as units of information, since the notion of a relation is a table

of data at a certain level. However, there are some problems associated with this method of assignment. For example, potential data overclassification, and inconsistencies caused when low-level relations are deleted in spite of there being high-level relations which reference their tuples (usually by a key value) (Ref 28:60). Even though these problems exist, they will not cause an unauthorized access.

Backup and recovery, in the revised model, are viewed as hidden objects since they can be performed automatically by the system without user awareness. The main purpose of the backup/recovery subsystem is to allow for the storing and retrieval of copies of data objects in a protected ARCHIVE (Ref 28:71). This archive is protected because only the backup and recovery processes are able to access it. This is accomplished by fixing the M(ARCHIVE) in the model, so the backup process has read only access. It is important to note that backup and recovery cannot operate as normal processes. In order to function, backup must run as system high-level if it is to observe all relations. Backup would then store the relations at the system high-level. In order to read these stored relations, recovery would also have to operate at the system high-level. Consequently, recovery has to be a trusted process in order to restore any relations that are classified at less than system high-level (Ref 28:72).

In summary, the following modifications, extensions, and additions are implemented in the revised model.

1. The simple security property is changed to use the subject's current security level rather than the maximum level in specifying non-discretionary authorization of observation.

2. The *-property is changed to use the subject's current security level rather than the levels of the accessed objects in specifying non-discretionary authorization of observation.

3. The organization of the directory objects is changed to offer more flexible and powerful capabilities.

4. The set of access modes is redefined to consist solely of observe and modify.

5. The set of requests is changed to reflect the change in model entities.

6. The security level is extended to include an integrity level, producing a protection level. Correspondingly, the security level functions (fs,fo,fc) are extended to produce (Hs,Ho,Hc). The simple protection and *'-properties follow.

7. The directory system is extended to allow searching for objects.

8. An object in the protected data management system is defined to consist of three parts: a permission matrix, a descriptor, and a value set.

9. Protection requirements are extended to apply to execute access.

10. The tranquility principle is extended to apply to subjects as well as objects.

11. Integrity levels, level functions, and properties are defined to be dual in nature to the security entities and are relevant to authorized modification control and not correctness or soundness.

12. The set of directory functions can be used to search directories.

13. The direct access function makes object access more flexible and is identified as the mechanism enforcing the protection requirements.

DISADVANTAGES

Normalization is the process of removing undesirable functional dependencies from the information stored in a relation (Ref 32). However, there are difficulties with the automatic normalization of relations in the revised model because functional dependencies must be defined apriori. As a result, automatic normalization is undesirable and therefore not included as part of the

revised model's relational approach. In addition, the revised model classifies only the relation as a whole instead of classifying each field (domain) of a relation. Bell and LaPadula's argument for doing this was based on the following: for convenience of observation and modification, all fields would have to be at the same level of classification anyway, and that this requirement is equivalent to placing a classification on the relation as a whole instead of on each field (Ref 1:267). Other research recommends that the classifications be attached to the fields of a relation (Ref 33). Consequently, more research work needs to be done in this area.

ADVANTAGES

Trusted objects are not included in the revised model since they are considered to be proven programs which are completely within the security perimeter and therefore not subject to the rules of the revised model.

The new directory system permits searching for all dominated objects which is very desirable in a general data management system (Ref 28:33). However, a change from the use of a unique path per object results in name uniqueness problems. The solution to this problem was to make an object's level part of its identification. Now the new directory system would accommodate the real-world requirement for allowing the identifier (existence) of an object to be stored in a directory lower than the actual object (essence) (Ref 28:37).

The accessing of an object, in the revised model, involves only one directory whereas in the original model an object could not be accessed until every directory object in the unique path to the object had been accessed. As a result, the ability to access directory objects is more flexible.

The incompatibility among hierarchies of objects was corrected in the revised model. This was accomplished through the new directory system structure.

In summary, the revised Bell-LaPadula model was the forerunner in implementing relational data bases in a database management system. In addition, it has corrected most of the disadvantages associated with the original model. Finally, both the original and the revised models have served as the basis for many design, prototype, and implementation efforts.

SUMMARY

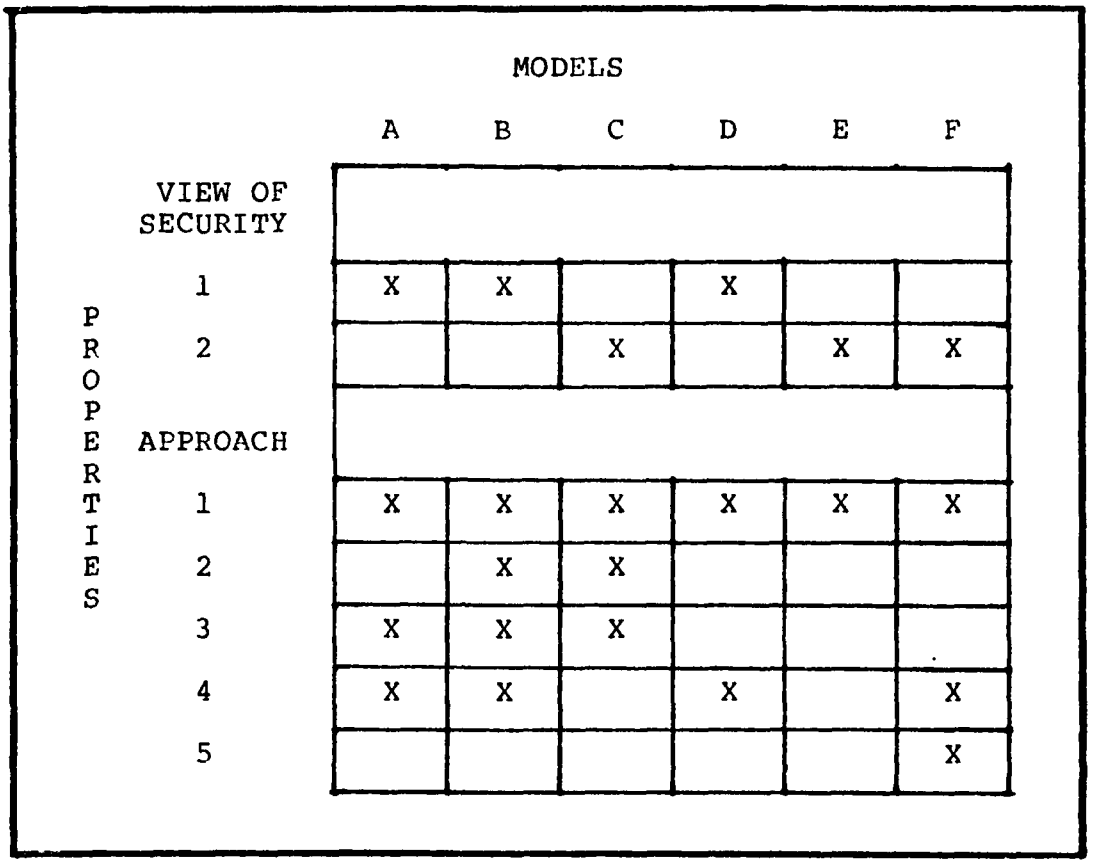


Figure 6: Comparison of Properties of Models

MODELS:
- A = Access Matrix
- B = UCLA Date Secure Unix
- C = Take-Grant
- D = High-Water Mark
- E = Bell-LaPadula (original)
- F = Bell-LaPadula (revised)

PROPERTIES:

View of Security;
- 1 = Models access to objects without regard to contents
- 2 = Models flow of information among objects

Approach;
- 1 = Model focuses on system structure (files, processes)
- 2 = Model focuses on operations on capabilities
- 3 = Model separates protection mechanism and security policy
- 4 = Systems based on or represented by this model have been implemented
- 5 = Model has been implemented using a relational database

Figure 6 compares the models with respect to approach, view of security, and use. To be useful, a comparison of the models should examine both the definition of security and the feasibility of implementing a computer system that performs the application required of it and can be verified to simulate the model. Unfortunately, such a comparison is difficult because few implementations based on these models have reached a stage where their performance can be reliably estimated or where verification of their security properties can be attempted (Ref 1:272). However, some of the models prove to be better candidates as bases for future secure computer systems than others.

Each model defines its own world and its own concept of security in that world. Thus a computer system that truly simulates any of the models will be secure in the sense defined by that model.

A problem common to all of the models is that they define security as absolute (an operation is either secure or not secure) (Ref 1:273). This approach does not help the designer or the implementer who must make trade-offs between security and performance.

With respect to their definitions of security, the models can be divided into two groups: those that are concerned only with controlling direct access to particular objects (Access Matrix model, the UCLA Data Secure Unix model, and the Take-Grant model); and those that are concerned with information flows among objects assigned to

security classes (original and revised Bell-LaPadula models). The High-Water Mark model falls between the two groups, since it is concerned with the security levels of the objects a process touches over time and only controls direct accesses to objects (Ref 1:273). Therefore, the appropriateness of a particular model will naturally depend on the application for which it is to be used.

IV MODEL REQUIREMENTS

This chapter discusses the requirements that should be implemented in the design of a CS/RS model for the AFIT DEL LSI-11 configuration using a relational database. However, due to the nature of the requirements, they are grouped into the following five categories for easy reference: General; Operating System; User; Data Base Administrator; and Relational Database. In addition, based on the requirements, this chapter discusses the rationale that was used in the selection of a computer security model (from those presented in Chapter III).

The DEL User established the five categories of requirements that should be used in the design of the CS/RS model.

GENERAL REQUIREMENTS

1. Since the Roth relational database was to be used as a pedagogical tool for teaching database management and manipulation techniques, a requirement for illustrating how modular design techniques could be used for implementing different security concepts was appropriate. In addition, the actual implementation of several security concepts should be demonstrated.

2. Any source code written for the CS/RS model should be well documented, modular, and easy to modify or replace if necessary. Also, the CS/RS source code should be independent of all other code written for the Roth

relational database so that any changes required in the CS/RS model will not ripple down to the database level.

OPERATING SYSTEM REQUIREMENTS

1. Any physical security measures implemented should prevent or minimize the loss by theft of equipment, diskettes, paper, and other external peripherals.

2. The operating system's directory file structure should require security measures to prevent unauthorized users from accessing or observing files when they are not authorized to do so. In addition, security restrictions are needed at the file level to prevent all the users from having the same ability to read, write, and execute a file.

3. Any backup and recovery procedures should automatically be done by the operating system.

USER REQUIREMENTS

1. Any security measures requiring IDs, passwords, classification levels, badges, keys or combinations thereof should have a minimum impact on the users.

2. There should be no "owner" of a relation or a tuple. However, one or more users should be able to assign an ownership-ID to a relation. NOTE: Any subsequent references to owner in this thesis will imply one or more users as having ownership. The ownership-ID should be used by the owner for assigning user access rights (read,

insert, delete, execute, and modify) to a relation. In addition, this ownership-ID should not be accessible by the DBA and should not resemble the IDs assigned to the users by the DBA for logging onto the DBMS.

3. Any unauthorized attempts by a user to access the relational database or the relations themselves should be logged and brought to the attention of the DBA or the owner, respectively. If possible, real time alarms should be implemented.

4. The relational database is to be on-line. Consequently, all the relations making up a relational database should be accessible by the Data Base Management System (DBMS).

5. Group access (for example, categories or classification levels) should be considered in addition to the individual user access rights assigned to a relation.

DATA BASE ADMINISTRATOR REQUIREMENTS

1. Only the DBA should have the ability to directly access the information in the Data Definition Language (DDL).

2. The DBA should be the only person who can grant authorized users permission to access the relational database (for example, a DBA assigned Identification (ID), password, and classification level).

3. The DBA should be responsible for creating, modifying, and deleting relational definitions and domain definitions.

4. Any information within any relation should not be accessible to the DBA unless access permission has previously been granted by the owner.

5. The DBA should not be allowed to enter, delete or issue access rights to a user requesting access to a relation. This job should be the responsibility of the owner. However, the DBA should be able to delete any relation with access rights assigned, upon owner request.

6. The DBA should be the only individual given permission to do the following:

(a) to enter information in order to define relations and domains;

(b) to receive an inventory listing of all domains and attributes; and

(c) to initialize domain and relation definition lists to empty.

RELATIONAL DATABASE REQUIREMENTS

1. Only authorized personnel should be allowed to access the relational database. In addition, the access rights (read, insert, delete, execute, and modify) granted to a user or a DBA should be enforced at the relation, tuple and attribute levels to prevent unauthorized access.

2. Security measures used to enforce the access

rights granted to a user should have a minimal impact on the relational database and the operating system.

3. Security measures should be implemented in order to differentiate between authorized and unauthorized access to a relation or an attribute. Then if any unauthorized access is attempted it can be detected and stopped.

4. The read, insert, delete, execute, and modify access rights should also be uniquely identified and protected in some fashion.

MODEL SELECTION

Based on the requirements previously defined in this chapter a CS/RS model was selected. By observing Figure 6 in Chapter III (page 74) it appeared that any one of the six models would be appropriate. Also mentioned within Chapter III was the fact that the models could be divided into two groups: those that are concerned only with controlling direct access to particular objects (Access Matrix model, the UCLA Data Secure Unix, and the Take-Grant model); and those that are concerned with information flows among objects assigned to security classes (original and revised Bell-LaPadula models). Since several of the requirements suggested protecting the flow of information from users or classes of users (objects) that are unauthorized to access data within a database, the second group of models seemed more likely to meet these requirements. In addition, the appropriateness of any

particular model depended on the application for which it was to be used. For purposes of multilevel security classification levels, those in the first group required the addition of security classification policies and the assessment of indirect information flows (for example, timing and storage channels) at the implementation level. However, the second group of models already met this requirement.

The formal verification of properties of system designs was a very important issue and an active research topic. Consequently, only the Bell-LaPadula models have been applied in more than one formally specified system. In regards to the High-Water-Mark, Access Matrix, and Take-Grant models the properties associated with each of these models are not stated in a form suitable for automated verification.

Another important concept relating to the Bell-LaPadula (revised) model is that it was the only computer security model that had been implemented using a relational database. Thus, many of the properties characterized by this model can be modified, if need be, and implemented into the design of the CS/RS model. In addition, most of the other models (Access Matrix, UCLA Data Secure Unix, and Take-Grant) separate the protection mechanisms and the security policies. However, Bell and LaPadula in their revised model demonstrated that this was not necessary because both integrity and security could be defined to be

dual in nature, thus avoiding any need for separate protection mechanisms and security policies.

Consequently, the CS/RS model designed was patterned after the revised Bell-LaPadula model. This was based on the following: the requirements previously defined in this chapter; the flexibility offered by the Bell-LaPadula revised model; and the fact that it was the only model that has been implemented using a relational database. However, many of the techniques used in the revised model can not be implemented because of the constraints imposed on the design by the LSI-11 and the Roth relational database. These constraints as well as other modifications needed for the CS/RS design are discussed in detail in Chapter V.

V CS/RS MODEL DESIGN

INTRODUCTION

This chapter discusses the CS/RS model, characteristics, components, and data structures that were used in the design of the CS/RS model. In addition, it addresses the model constraints that were encountered during the design.

CS/RS MODEL

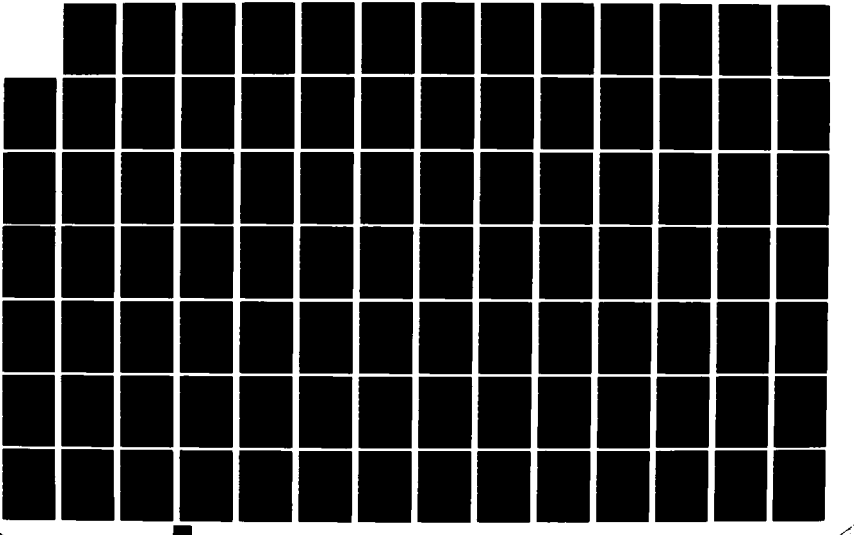
The CS/RS model was divided into the following three areas: the DBMS Access Monitor; the Relation Access Control List; and the Relation Access Monitor.

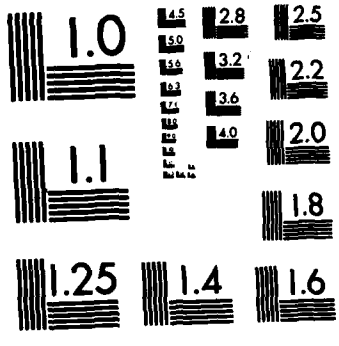
The DBMS Access Monitor was designed to prevent unauthorized users from gaining access to the DBMS. This was accomplished by assigning authorized users a unique user-id and password which would be used for logging onto the DBMS. Once the user-id and password had been entered, they would be passed by the software to the DBMS Access Monitor. Then the DBMS Access Monitor would verify the logon attempt by comparing the user-id and password entered to the tuples contained in a permanent relation called ACCESS. These tuples would consist of an ID, password, user-name, and classification level. In addition, only those users having access to the DBMS would be contained in this relation. If a match was found then the user would be granted permission to access the DBMS otherwise permission would be denied.

The Relation Access Control List was designed as a binary tree for storing the following information: the relation names defined in the DDL; the owner-ids; and the lists of user-ids with their assigned access rights. In addition, the owner of a relation would be responsible for entering this information.

In order to enter information into the Relational Access Control List the owner of a relation would go through the normal logon procedures. Once permission to access the DBMS had been granted the owner would call up the Relation Access Control List module. Then the owner would enter the relation name and a unique, owner defined owner-id. After this information had been entered the owner would need to enter the user's name (first name, blank, middle initial, blank, last name) and the access rights (read, insert, delete, modify, or execute) granted to this user. It was necessary to enter the user's name in order to prevent any compromising of user-ids. Once the user's name and access rights had been entered, the user's name would be passed by the software to the DBMS Access Monitor which would retrieve the tuple from the ACCESS relation matching this user's name. If the tuple was found then the user-id would be retrieved and passed back by the software to the Relation Access Control List module. Then the user-id and access rights granted to this user would be entered into the list of users with their associated access rights. It is important to note

AD-A124 840 THE ANALYSIS AND DESIGN OF A COMPUTER SECURITY/RECOVERY 2/3
SYSTEM FOR A RELA.. (U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W P REED
UNCLASSIFIED 17 DEC 82 AFIT/GCS/EE/82D-26 F/G 9/2 NL





that only the user-ids and access rights may be added, deleted, or modified in the Relation Access Control List.

The Relation Access Monitor was designed for verifying whether a user had been granted access (read, insert, delete, modify, or execute) to a relation. If access had been granted to a relation then each attribute's classification level would be checked to ensure that the user's classification level (which was retrieved from the ACCESS relation at logon) was greater than or equal to the classification level of the attribute. If not, access to this attribute would be denied. It is important to note that classification levels were only assigned to the attributes and not to the relation or to the tuples.

CS/RS MODEL CHARACTERISTICS

RELATION AND DOMAIN CHARACTERISTICS

The DDL system is accessible only by the DBA for entering domain and relation definitions. The procedures used for defining domains and relations are identical to those used by Mark A. Roth (Ref 6) and Linda M. Rogers (Appendix A) (Ref 34) with the exception of two modifications to the relation definition.

In order to define a relation the owner must provide the DBA with the following information:

1. RELATION NAME => unique to the relational data base and consisting of 1 to 15 characters

2. ATTRIBUTE NAME/CLASSIFICATION LEVEL/DOMAIN NAME TRIPLES =>

(a) Attribute names must be unique within a relation

(b) Classification level must be either a 1, 2, 3, 4, or 5 with each higher classification level having access to all lower classification levels. That is, $1 < 2 < 3 < 4 < 5$.

(c) Domain names must have been previously defined in a DDL domain definition according to the following rules:

(1) DOMAIN NAME => unique to the relation being defined and consisting of 1 to 80 nonblank characters; and

(2) DOMAIN TYPE => a single character represented by either a (C) character, (T) text, (R) real, or (I) integer where

(a) R => the maximum number of digits before a decimal point and the maximum number of digits after a decimal point

(b) C,T,I => the maximum number of characters or digits

(d) Sorted or unsorted directory for each attribute

3. PRIMARY KEYS =>represented by previously defined attributes

4. CONSTRAINTS => attribute specified constraints consisting of >, <, or =.

The two modifications that were made in order to define a relation were: (1) the addition of the classification level to the attribute name/domain name pair; and (2) the deletion of security passwords for the user ID and the read, delete, modify, and insert access rights.

The DBA is the only individual who can enter the information into the domain and relation definitions. Consequently, it is very important that the information forwarded to the DBA by the owner is accurate and complete.

RELATION ACCESS RIGHTS

In order to access a relation or to manipulate the tuples within a relation the following five access rights will need to be defined: Execute; Read; Insert; Modify; and Delete. It is important to note that the read, insert, modify, and delete access rights are associated with the editing of a relation. On the other hand, the execute access right is only associated with the execution of a Command File. In addition, the owner will have permission to use all five access rights, whereas other users will only be allowed to possess those access rights assigned by the owner.

1. EXECUTE (E) => will allow the user access to a relation when executing a Command File containing queries. This access right is associated only with the execution of a Command File.

2. READ (R) => will allow the user to observe the attribute values of a tuple on the CRT screen provided the classification level of the user is greater than or equal to the attribute classification level. This access right

is associated only with the editing of a relation.

3. INSERT (I) => will allow the user to create a new tuple of attribute values within a relation. This access right is associated only with the editing of a relation.

4. MODIFY (M) => will allow the user to modify the attribute values of a tuple within a relation. This access right is associated only with the editing of a relation.

5. DELETE (D) => will allow the user to delete a tuple of attribute values within a relation. This access right is associated only with the editing of a relation.

CS/RS MODEL COMPONENTS

DBMS ACCESS MONITOR

In order to access the DBMS, a DBA-assigned user ID and password will have to be entered. This ID and password will be verified to determine whether a particular user is to be granted access to the DBMS. If an invalid user ID or password is entered, access to the DBMS will be denied and the user ID and password will be logged into a file for use by the DBA. Each ID will be nine digits long with each digit assuming a value from 0 thru 9. By observing these two ID constraints it can be shown that the 10 values per digit (0 thru 9) and the 9 digits per ID combination represent 10 to the 9th or one

billion ID combinations. In addition, each password will consist of nine nonblank characters, with each character being represented by a number from 0 to 9 or a letter from A to Z. Since the password may contain a combination of both numbers and letters it can be shown that the 36 values per character (0 thru 9 and A thru Z) and the 9 characters per password represent 36 to the 9th or 1.0156 trillion password combinations. Consequently, there would be a possible 1.0156 to the 23rd possible ID/password combinations. Since fewer than 100 users will access the DBMS, the chances of an intruder randomly selecting a valid ID/password combination and gaining access to the DBMS are quite small. Therefore, based on the number of users to have access to the DBMS, the 1.0156 to the 23rd ID/password combinations should provide an adequate measure of security. Associated with each user ID will be a DBA-assigned classification level. Regarding the AFIT DEL requirements, five classification levels should be sufficient. This was based on the following groupings: Students; Teachers; Secretaries; Laboratory Technicians; and Administrators. Each classification level will be a single digit, represented by a value from 1 to 5. In addition, each higher classification will have access to all lower classification levels. That is, $1 < 2 < 3 < 4 < 5$.

RELATION ACCESS CONTROL LIST

Once the DBA has entered the domain and the relation definitions into the DDL, it will then be the responsiblity of the relation's owner to enter appropriate information into the Relation Access Control List.

The information to be entered will consist of the relation name defined in the DDL, the OWNERID, and the list of user IDs with their associated access rights. The relation name will consist of 1 to 15 nonblank characters as defined by the DDL. The OWNERID will consist of 10 nonblank characters, with each character being represented by a number from 0 thru 9 or a letter from A to Z. Since this ID may contain a combination of both numbers and letters it can be shown that the 36 values per character (0 thru 9 and A thru Z) and the 10 characters per OWNERID combination represent 36 to the 10th or 3.6562 quadrillion combinations. Consequently, the chances of an intruder randomly selecting a valid OWNERID for a particular relation is very, very minute. It was necessary to increase the size of the OWNERID in order to maintain a high level of security for protecting the Relation Access Control Lists. The 3.6562 quadrillion possible combinations support this claim. The OWNERID will not exist in the DDL and will not be accessible to the DBA. Both the relation name and the OWNERID will need to be entered before any user IDs can be assigned to a relation. Once this has been accomplished, the owner will enter a user's name (first name, blank, middle initial, blank,

last name) and the access rights to be granted to this user. As previously mentioned, the access rights consist of read, delete, modify, execute, and insert. Consequently, any combination of these access rights may be assigned to a user.

In order to prevent any compromising of the DBA assigned user-ids it was necessary to prevent the owners of the relations from having access to them. This was accomplished by storing the DBA assigned user-ids, passwords, user names, and classification levels in a permanent relation defined in the DDL. This is the same relation that was used by the DBMS Access Monitor for verifying logon access. Thus, if the owner of a relation wished to grant users access to this relation, it would only be necessary to enter the users' names. Once a user's name had been entered, it would be passed by the software to the DBMS Access Monitor. Then the DBMS Access Monitor would fine the tuple within the permanent relation (previously mentioned) associated with this user's name and retrieve the user-id. This user-id would be passed back by the software to the Relation Access Control List where it would be entered into the list of user-ids with associated access rights for this relation.

In order to prevent the relation names from being deleted by unauthorized users it was necessary to grant only the DBA permission to do this. If the name of a relation is to be deleted, the owner will make a request

in writing to the DBA. Once this request has been made, the DBA will access the DDL program (which is only accessible to the DBA) and enter the name of the relation to be deleted. Consequently, all the information pertaining to this relation (relation name, OWNERID, and list of users with associated accesss rights) will be deleted from the Relation Access Control List. Clearly, such information should not be deleted by the DBA without owner approval.

It is important to note that the owners of the relations will have permission to do the following:

1. Add the relation name and ownership-ID to the Relation Access Control List. This procedure is done automatically by the software and is done only once. Consequently, if the owner of a relation wishes to change the ownership-id the following will need to be done by the owner:

a) Request in writing that the DBA remove this relation name from the Relation Access Control List; and

b) Reenter the relation name and new ownership-id, as well as the user names and the access rights to be granted to these users;

It should be noted that if the owner of a relation requests to change the ownership-id, all the information (relation name, OWNERID, and list of user-ids with associated access rights) associated with this relation name is removed from the Relation Access Control List;

2. Add user names and/or access rights to an owner relation defined in the Relation Access Control List;

3. Delete user names and/or access rights to an owner relation defined in the Relation Access Control List.

4. Replace user names and/or access rights to an owner relation defined in the Relation Access Control List.

RELATION ACCESS MONITOR

Most users will desire to execute a Command File containing relation and attribute queries. Therefore, it will be necessary to ensure that only those users having access to a relation and its attributes are allowed to do so. Consequently, whenever a user logs into the DBMS with the DBA-assigned user ID and password, it will be verified by the DBMS access monitor. If it is a valid user, the ID and the classification level associated with the ID will be saved and passed on to the relation access monitor. When the relation access monitor executes the Commmand File it will access the relation access control list to verify that the user has (E) access to the relations specified in the Command File. If access has been granted to a relation each attribute's classification level will then be checked to ensure that the user's classification level is greater than or equal to the classification level of the attribute. If not, access to this attribute's information will be denied and only those attributes the user has access to will be manipulated by the Command File queries. Consequently, the relation created by the Command File queries will only contain tuples with attributes values having classification levels less than or equal to the classification level of the user. If a user does not have access to a relation, the user's ID and

the relation to which access has been denied will be logged into a file for use by the DBA and any further execution of the Command File will be terminated.

In a similar fashion the Relation Access Monitor will be used by the editor. Again the Relation Access Monitor will be executed and it will access the Relation Access Control List to verify that the user has (R), (I), (D), or (M) access to the relation to be manipulated. For instance, if a user wishes to insert tuples into a relation the (I)nsert option is selected. If the user has been granted the (I) access right to this relation, then each attribute's classification level is checked. If the user's classification level is greater than or equal to the classification level of the attribute, permission to enter a value into the attribute will be granted otherwise it will be denied. On the other hand, if a user tries to insert into a relation to which access has not been granted, then the user's ID and the relation to which access has been denied will be logged into the same file, previously mentioned, for use by the DBA. This same set of procedures applies for the (D)elete, (M)odify, (C)opy, and (T)ransfer edit options.

CS/RS MODEL DATA STRUCTURES

DBMS ACCESS MONITOR

Since the CS/RS model was to be implemented using a relational database, the decision was made to actually use

a relation for storing the ID, password, user name, and classification level of all the individuals that would have access to the DBMS. The main reason for choosing the relational concept was that it would save a lot of core memory compared to a matrix, linked list, or tree structure which would require all of the users to reside in core memory. To ensure security, the DBA would be the only individual who could add, delete, insert, modify or observe any of the attribute values.

The name of the relation would be ACCESS and it would contain the following attributes: ID, IDPSW, NAME, and CLASS. Figure 7 illustrates what the tuples in the ACCESS relation might look like.

ACCESS RELATION

ID	IDPSW	NAME	CLASS
123456789	ABCDEFGHI	WILLIAM REED	1
482111110	A23BCH50A	TOM HARTRUM	2
.	.	.	.
.	.	.	.
.	.	.	.

FIGURE 7: Access Relation

If a user tried to log on, the ID and password entered would be retrieved and passed to the DBMS Access

Monitor. At this time the following command string would be executed.

```
SELECT ALL FROM ACCESS WHERE ID = 123456789 AND
IDPSW = ABCDEFGHI GIVING TEMP_REL
```

Once this command string had been executed, a new relation called TEMP_REL would be created. This new relation would contain no more than one tuple and would pose no security threat since the information in the tuple pertains to the individual trying to log on. Figure 8 illustrates what a single tuple in the TEMP_REL relation might look like.

TEMP_REL RELATION

ID	IDPSW	NAME	CLASS
123456789	ABCDEFGHI	WILLIAM REED	1

FIGURE 8: Temp_rel Relation

If the ID and password entered at logon match the ID and IDPSW attribute values contained in the TEMP_REL relation tuple then access to the DBMS would be granted, if not access would be denied. In either case the log on attemp would be entered into the file ERROR.DATA for use by the DBA.

RELATION ACCESS CONTROL LIST

The information to be stored in the Relation Access Control List consists of a relation name defined in the DDL, the OWNERID, and the list of user IDs with their associated access rights. In order to store this information, five data structures were considered. One involved a relation, two involved linked list matrices and two involved tree structures. Each data structure is discussed below.

RELATION

This method would require defining a separate relation for each relation defined in the DDL. This relation would be made up of tuples containing a user name and/or user-id and the access rights granted to this user. A second possibility would be to define a separate relation for each user-id and/or user name. This relation would consist of tuples containing the relation name and the access rights to this relation. However, either method would pose some problems. First, who should be allowed to update the relations as well as who is going to do the updating. Second, relations defined in the DDL would increase significantly. This would require more space or memory which is already at a premium. Third, for each relation defined in the DDL a similar relation would have to be defined or updated. Again space and who should and who will do the updating is of importance. Finally,

the concept of a relation owner monitoring who has access to the relation becomes much more difficult to implement and keep track of. As a result, either one of these approaches could evolve into a vicious circle if not implemented carefully.

MATRIX

Method one is often called an access control list because it involves storing the matrix by column i.e. relation names (objects). This method's major advantage is that the Relation Access Control List need not be available in fast-access memory since a particular object may not be active often enought. Consequently, this method may be a practical solution.

Method two is often called a capability list because it involves storing the matrix by rows i.e. user-ids (subjects). The major advantage with this method is that it is more efficient because the capability lists must reside in fast-access memory whenever the subject is active. However, since fast-access memory is limited on the LSI this method would probably not be feasible.

TREES

Method 1: Similar to the access control list implementaion. However, the relation names would be associated with the nodes of a tree and the user-ids and access rights would be attached to these nodes.

Method 2: Similar to the capability list implementation. However, each node of the tree would now contain the user-id with the relation names and access rights to the relation attached to these nodes.

The relational method of storing the Relation Access Control Lists seems to be the worst approach based on the problems previously mentioned. The two matrix data structures would require less storage because only a single pointer would be required to connect the nodes, while the binary tree structures would require more storage because each node has two pointers. However, assuming the binary tree is created in such a way so that half the nodes appear on the left side of the tree and half appear on the right side of the tree, the average time required in searching for a particular node would be considerably less than the time required to search a linearly, linked list. For instance, the worst case time required to search the binary tree would be of order log n (where n equals the number of nodes in the binary tree) whereas the worst case time for searching the linearly, linked list would be of order n (where n equals the number of nodes in the linked list). Since many searches will be required and the time factor is of critical importance to the user, the binary tree structure is more appropriate. Thus, considering the two methods previously discussed under tree structures, Method 1 of storing the relation names in the nodes and attaching the user-ids and access

rights to each node is more appropriate. The reason for choosing this method is that it does not separate the user-id from the access rights granted by the owner. In addition, assuming there are more users than there are relation names, the time required to search for a particular node in the binary tree would be considerably less.

RELATION ACCESS MONITOR

In order to verify what rights a particular user has to a relation it was necessary to retrieve the access rights from the Relation Access Control List. Once the access rights were retrieved, they were stored in a five byte, fixed length array where each byte of the array would be compared to the literals 'R', 'D', 'I', 'M', or 'E' (Read, Delete, Insert, Modify, and Execute). If a match was found then the particular flag associated with this access right would be set to true, otherwise it would be set to false. Consequently, if a user wanted to insert tuples into a relation and the access rights had been granted, the INS_FLAG would be set to true and permission would be granted provided the classification level of the attributes
to be inserted were less than or equal to the classification level granted to the user.

CS/RS MODEL CONSTRAINTS

Several constraints were imposed on the CS/RS model

by the LSI-11 and the Roth relational database. For example, many of the security measures implemented in the Bell-LaPadula (revised) model were enforced at the operating system level. However, the UCSD Pascal Operating System did not provide this luxury. Consequently, many of the following security measures could not be implemented:

1) A set of directory functions which is used by the Bell-LaPadula model in searching for directories and for objects;

2) A set of access modes consisting solely of observe and modify;

3) The implementation of integrity levels to ensure any additions, modifications, or deletions to a relation do not jeopardize other relations in the DBMS; and

4) The use of the direct access function for enforcing the protection requirements.

A second and very important constraint was that the available core memory on the LSI-11 was too limiting. Since most of the Roth relational database code had previously been segmented, it was oftentimes difficult to work in the necessary CS/RS modules without resegmenting. Another LSI-11 constraint involved backup and recovery procedures. Currently, this is almost impossible to do under the UCSD Pascal Operating System. However, backup and recovery procedures can be written to interface with the Roth Relation Database System. These procedures should be performed automatically by the DBMS without user intervention. Finally, compiling and linking of the

binary code file into an executable module took an enormous amount of time, especially when segmentation was involved.

VI CS/RC MODEL INTEGRATION AND TESTING

The User's Guide for the Data Definition (DDL) Facility can be found in Appendix A. It includes all the CS/RS model changes that are relevant to the DDL Facility. The User's Guide for the the Roth Relational Database System can be found in Appendix B. It includes all the CS/RS model changes that were integrated into the Roth Relational Database System. Finally, Appendix C shows all of the structure charts that were used in the design of the CS/RS model.

The original relational DBMS was conceived by Mark A. Roth (Ref 6) in a 1979 thesis. Then in 1982, Linda M. Rogers (Ref 34) separated the DBMS into the Data Definition Facility and the Roth Relational Database System. In addition, executable baseline models were developed for each. These baseline models were used as the basis for implementing the CS/RS model designed. Figure 9 represents the Data Definition Facility structure chart with any new or modified modules represented by an "*". Figure 10 represents the Roth Relational Database System structure chart with any new or modified modules represented by an "*".

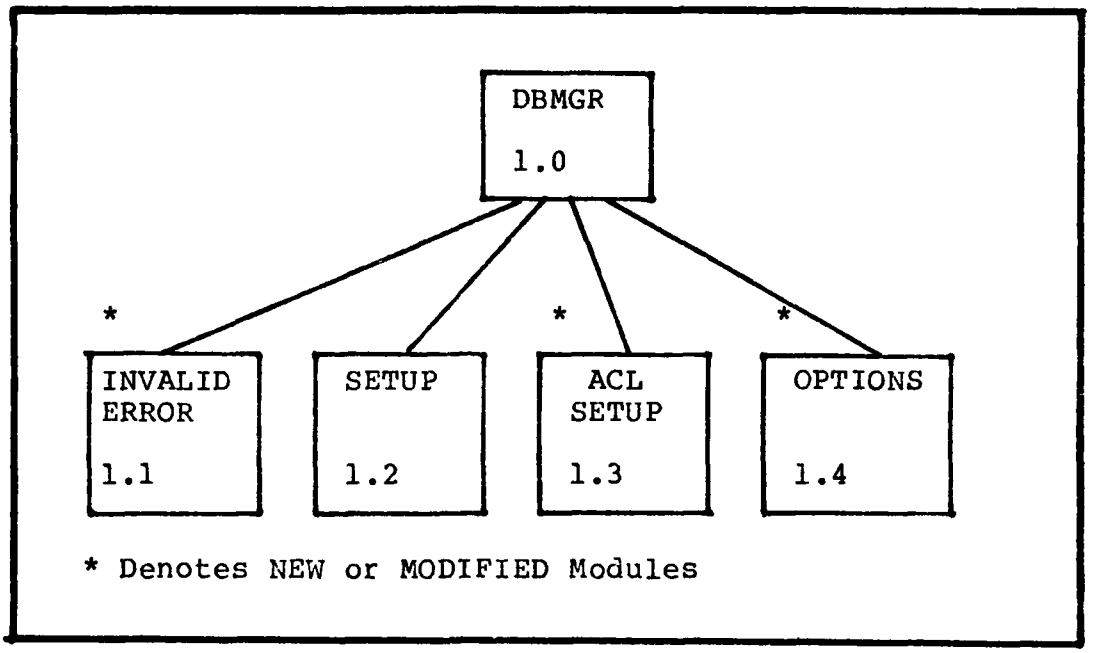


FIGURE 9: Data Definition Facility Structure Chart

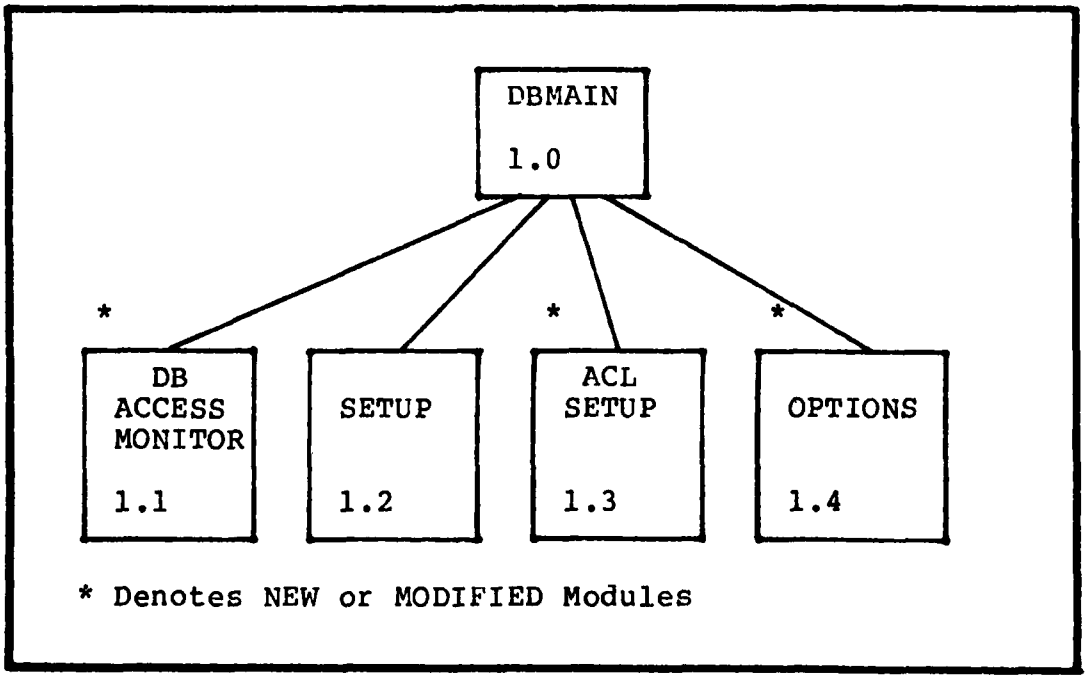


FIGURE 10: Roth Relational Database Structure Chart

Due to time constraints, none of the CS/RS modules

were tested. However, all the CS/RS modules designed were coded, implemented, and shown to be free from compile errors. The next logical step in this thesis would have been to link the binary code files, created by the compiler, into an executable module for testing. Once this had been accomplished the followng test procedures would have been attempted.

1. Test the DDL Facility to ensure that the new relational definitions contained the changes outlined in Appendix A.

2. Test the log on procedures to ensure that they worked. Currently, there is no way to retrieve information from a relation. Therefore, in the EXTRACT module (1.1.2.2), code was included that would prompt the user for an ID, password, and classification. The ID and password entered would have to match that which the tester logged on with.

3. The ACL_SETUP module (1.3) would need to be tested next to ensure that the file ACLSET.DATA could be read. If information existed on this file, it would be read and stored into a binary tree in memory. Next this tree would have to be traversed in PREORDER format, printing out each node, to verify that the tree was built correctly.

4. Then the UPDATE module (1.4.1) would be tested.

This would require the owner of a relation to enter the name of a relation defined in the DDL and an OWNERID. If this relation name was valid, a search of the Access Control Tree (ACT) (1.4.1.3) would be attempted. If the relation name was not in the ACT, the ADD_REL_OWNERID (1.4.1.5) would add it. Then the owner would be required to enter the user's name and the access rights to be assigned. Again since there is no way to retrieve the USERID from the ACCESS relation, code was implemented that would allow the owner or tester to type in an ID to associate with the user's name. Finally, the owner would select an option (Add, Delete, or Modify). Each of these options would have to verified to ensure that they work. This can be accomplished by printing out the ACT after each option has been attempted.

5. The QUIT module (1.4.4) would be tested next to ensure that the ACT created by the UPDATE module (1.4.1) was written correctly to the ACLSET.DATA file. This can be verified by printing out the ACT and comparing it to what was written to the file.

6. The INVALID_ERROR module would be tested to ensure that each of the error codes that were defined in Appendix B worked properly. This would require finding the modules that create the error and then generate the error condition. Each error condition can be verified by accessing the ERROR.DATA file.

7. Finally, the VALID INSERT COMMAND (1.4.2.1), the VALID DELETE COMMAND (1.4.2.2), and the VALID MODIFY COMMAND (1.4.2.3) modules need to be tested to ensure that the VALIDID and CK_ACCESS_RIGHTS modules are working properly. Verification can be made by observing the tuples of a relation created by each command module.

One final comment, is that there exists one more module, EXECUTE (1.4.6.4) that needs to be tested using the VALIDID and CK_ACCESS_RIGHTS modules. However, this module has not been implemented into the Roth Relational Database System.

VII CONCLUSION

OVERVIEW

The title of this thesis implies that two goals were sought. One was to analyze several computer security models and the other was to design a CS/RS model for a Relational Database Management System. The first goal was met, however it was quite time consuming. A decision was made to summarize the computer security models and to list the advantages and disadvantages associated with each.

The second goal was achieved in that a CS/RS model was designed from those security models analyzed for the Roth Relational Database System. However, the integration and testing of this model has yet to be accomplished. Since the Roth Relational Database was to be used in a pedagogical environment, the security measures implemented in the model will prove to be very beneficial. In addition, a tremendous amount of flexibility was available in selecting the actual design of the CS/RS model. This was primarily due to the fact that very few of the computer security models analyzed had ever been implemented using a relational database.

FUTURE RECOMMENDATIONS

The initial scope of this thesis was to actually design, code, implement, and test a CS/RS model using a relational database. However, due to time constraints only the design, module coding, and module interface to the Roth

Relational Database was accomplished. Because the CS/RS model was modularly designed, the actual integration and testing of the model can be accomplished through a special study project.

The following recommendations are given concerning what specifically needs further work or investigation.

(1) Implement the appropriate CS/RS modules into the execute command file module and the display module which have yet to be designed and coded.

(2) Look in to the possibility of incorporating the CS/RS model into the operating system.

(3) Improve the security measures for storing diskettes, files on diskettes, and operating system access methods.

(4) Implement a full backup and recovery capability.

(5) Establish a set of performance evaluation tools for measuring, fine-tuning, and comparing the data structures used in the CS/RS model.

(6) Transition the entire Roth Relational Database Management System to an alternative system architecture that would use an alternative high-level language such as PL/1, C or ADA.

(7) Consider the feasibility of implementing the classification levels at the relation and tuple levels.

FINAL COMMENT

Relational databases promise to provide a simple and flexible approach to a person or business's information retrieval needs. However, they require looking at security measures from a different perspective when compared to other types of databases. Consequently, this thesis effort was an attempt to take some of those security measures and design them into a viable CS/RS model that could be used in a Relational Database Management System.

BIBLIOGRAPHY

1. Landwehr, Carl E., "Formal Models for Computer Security", Computing Surveys, Vol. 13, September 1981.

2. Clark, David D. and David D. Redell, "Protection of Information In Computer Systems",Compcon 75 (Fall 75), IEEE Catalog No. 75CH1050-14. IEEE Computer Society, Long Beach, California.

3. Hartrum, T., Development Plan For The DEL LSI-11 Network/ System, AFIT/ENG inhouse document, February 1982.

4. Software Product Description, Digital Research, August 1981. (Available from AFIT/ENG).

5. UCSD (Mini-Micro Computer) PASCAL, Version II.0, Institute for Information Systems, Unversity of California, San Diego (March 1979). (Available from AFIT/ENA).

6. Roth, Mark A., The Design and Implementation of a Pedagogical Relational Database System, Thesis, 15 December 1979. (Available from AFIT/ENA).

7. Lampson, B. W., "Protection", in Proc. Fifth Princeton Symposium on Information Sciences and Systems, Princeton University, March 1971, pp. 437-443, reprinted in Operation Systems Review, Vol. 8, No. 1, January 1974, pp. 18-24.

8. Anderson, J. P., "Computer Security Technology Planning Study", ESD-TR-73-51. Vol. 1, ESD/AFSC, Hanscom AFB, Bendford, Massachuetts, October, 1972.

9. Dennis, J. B. and E. C. VanHorn, "Programming Semantics for Multiprogrammed Computations", Communications ACM, Vol. 9, No. 3, March, 1966.

10. Lampson, B. W., "Dynamic Protection Structures", AFIPS Conference Proceedings 35 Fall Joint Computer Conference 1969, pp. 27-38.

11. Graham, G. S. and Peter J. Denning, "Protection - Principles and Practice", in Proceedings 1972 AFIPS Spring Joint Computer Conference, Vol. 40, AFIPS Press, Arlington, Virginia, pp. 417-427.

12. Scientific Honeyweller, Vol. 2, No. 2, June 1981.

13. Anderson, J. P., "Computer Security Technology Planning Study", ESD-TR-73-51, Vol. 1, ESD/AFSC, Hanscom AFB, Bedford, Massachuetts, October 1972.

14. Popek, Gerald J., Mark Kampe, Charles S. Kline, Allen Stoughton, Michael Urban and Evelyn J. Walton. "UCLA Secure Unix", Proceeding AFIPS National Computer Conference, Vol. 48, AFIPS Press, Arlington, Virginia, 1979, pp. 355-364.

15. Walker, Bruce J., Richard A. Kemmerer and Gerald J. Popek. "Specification and Verification of the UCLA Unix Security Kernel", Communications ACM, Vol. 23, No. 2, February, 1980, pp. 118-131.

16. Wulf, W. A., R. L. London and M. Shaw. "An Introduction to the Construction and Verification of ALPHARD Programs", IEEE Transformation Software Engineering, Vol. 2, No. 4, December, 1976, pp. 253-265.

17. Kampe, Mark, Charles Kline, Gerald J. Popek and E. Walton. "The UCLA Data Secure Operating System", Technical Report, University of California, Los Angeles, July, 1977.

18. Popek, Gerald J. "Protection Structures", Computer, Vol. 7, No. 6, June, 1974, pp. 22-33.

19. Kemmerer, Richard A. "Verification of the UCLA Security Kernel: Abstract Model, Mapping, Theorem Generation and Proof", Doctor of Philospophy Thesis, UCLA, Los Angeles, California, 1979.

20. Ritchie, D. M. and K. Thompson. "The Unix Time-sharing System", Bell System Technical Journal, Vol. 57, No. 6, July-August, 1978.

21. Jones, Anita K. "Protection Mechanism Models: Their Usefulness", Foundations of Secure Computations, Academic Press, New York, 1978, pp. 237-254.

22. Bishop, Matt and Lawrence Snyder. "The Transfer of Information and Authority in a Protection System", Proceedings 7th Symposism Operating Systems Principles, ACM SIGOPS Operating Systems Review, Vol. 13, No. 4, December, 1979, pp. 45-54.

23. Snyder, Lawrence. "On the Synthesis and Analysis of Protection Systems", Proceedings of 6th Symposism on Operating Systems Principles, ACM SIGOPS Operating Systems Review, Vol. 11, No. 5, November, 1977, pp. 141-150.

24. Harrison, Michael A., Walter L. Ruzzo and Jeffrey D. Ullman. "Protection in Operating Systems", Communications ACM, Vol. 19, No. 8, August, 1976, pp. 461-471.

25. Jones, Anita K., R. J. Lipton and Lawrence Snyder. "A

Linear Time Algorithm for Deciding Subject-Object Security", Proceedings of 17th Annual Foundations Computer Science Conference, Huston, Texas, 1976, pp. 33-41.

26. Fonden, Robert W. Design and Implementation of a Backend Multiple-Processor Relational Data Base Computer System, AFIT Thesis, December, 1981. (Available from AFIT/ENE).

27. Bell, D. E. and Leonard J. LaPadula, "Secure Computer System: Unified Exposition and Multics Interpretation", MTR-2997, MITRE Corporation, Bedford, Massachuetts, July, 1975, pp. 5-128.

28. Grohn, Michael J., "A Model of a Protected Data Management System", ESD-TR-76-289, Electronic Systems Division, Air Force Systems Command, Hanscom, Air Force Base, Bedford, Massachuetts, June, 1976, pp. 1-118.

29. Millen, J. K., "Security Kernel Validation in Practice", Communications of the ACM, Vol. 19, No. 5, May, 1976, pp. 244-245.

30. Bell, D. E., Secure Computer Systems: A Refinement of the Mathematical Model, ESD-TR-73-278, Vol. 3, The MITRE Corporation, Bedford, Massachuetts, April, 1974, p. 18.

31. Codd, E. F. and C. J. Date, Interactive Support for Non-Programmers: The Relational and Network Approaches, IBM Research Report RJ1400, San Jose, California, June, 1974 p. 4.

32. Codd, E. F., Normalized Data Base Structure: A Brief Tutorial, IBM Research Report RJ935, San Jose, California, November 1971.

33. Hinke, Thomas H. and Marvin Schaefer, "Secure Data Management System", RADC-TR-75-266, Rome Air Development Center, Air Force Systems Command, Griffiss, Air Force Base, New York, November, 1975.

34. Rogers, Linda M., The Continued Design and Implementation of the AFIT Relational Database System, Thesis, 17 December 1982, (Available from AFIT/ENA).

AFIT/GCS/EE/82-4

USERS'S GUIDE FOR THE AFIT RELATIONAL DATABASE SYSTEM (THE DATA DEFINITION FACILITY)

APPENDIX A

TO

THE ANALYSIS AND DESIGN
OF A
COMPUTER SECURITY/RECOVERY SYSTEM
FOR A
RELATIONAL DATABASE MANAGEMENT SYSTEM
THESIS

BY

WILLIAM P. REED
CAPT USAF

Graduate Information Systems
17 December 1982

CONTENTS

I. Introduction and Overview. 117

The Data Definition Facility: An Overview. 117

II. Format, Key Words, and Names. 119

III. How to Use the DDL Facility 121

Starting the System 121
The Logon Procedure 121
The Outermost Level Commands. 124

Inventory 124
Initialize. 126
Remrelation 126
Define. 127

Innermost Define Commands 127

Define Domains. 127
Define a New Relation 129

IV. Command Summary 131

Define. 131

Define Domains. 131
Define a New Relation 132
Input from Mass Storage 132
Output to Mass Storage. 133

Inventory 133
Initialize. 133
Remrelation 133

V. Error Definitions 133

Overview 133
Error Defined 134

VI. Changing the System 137

Overview. 137
Bringing Up the Data Definition Facility. 140

APPENDIX A

USER'S GUIDE FOR
THE AFIT RELATIONAL DATABASE SYSTEM
THE DATA DEFINITION FACILITY

INTRODUCTION AND OVERVIEW * SECTION 1

The Data Definition Facility (DDL Facility) described in this document is a program intended to run on a stand alone minicomputer under the control of the UCSD Pascal Operating System, Version II.0. All software is written in Pascal, resulting in relatively straightforward maintenance and enhancement.

This program is designed to be used primarily with a CRT terminal as the CONSOLE device.

The function of the DDL Facility is to provide the Data Base Administrator with the capability to define domain and relation definitions. This program only allows the DBA to construct domain and relation definitions. In addition, the DDL Facility allows the DBA to obtain a full listing of the domain and relation definitions as well as the capability to destroy these definitions. Finally, the DDL Facility allows only the DBA to remove relation names and associated information from the Access Control Tree (ACT). Clearly, such information should not be deleted by the DBA without owner approval.

1.1 The Data Definition Facility: An Overview

The structure of the data definition facility is best conceptualized in terms of the "tree-like" structure diagram

in Figure A-1.

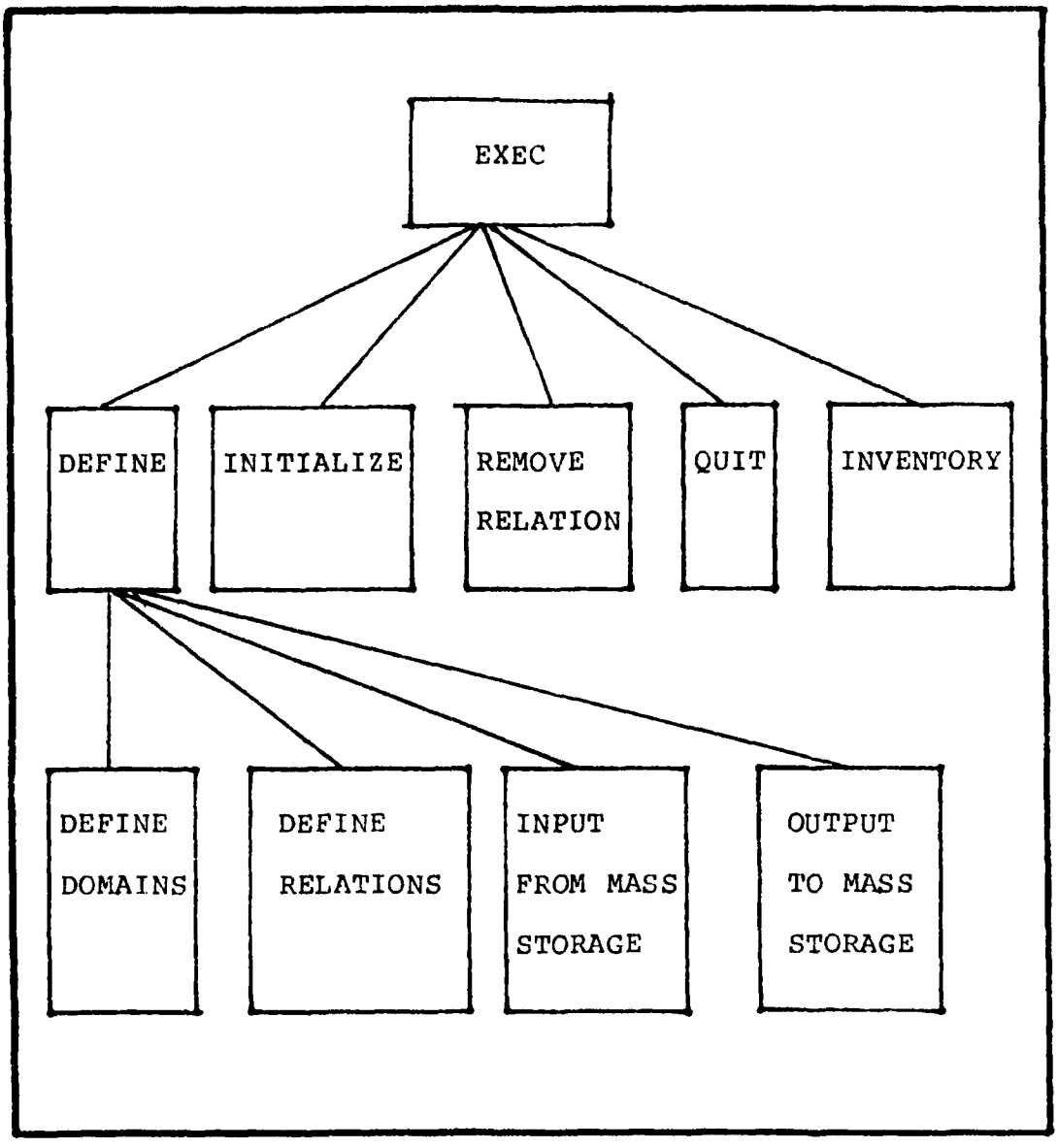


Figure A-1. The Data Definition Facility

While the DBA is in a particular level, the facility displays a list of available commands called the "prompt-line." The prompt-line will usually appear at the top of the screen. Commands are usually invoked by typing a single character from the CONSOLE device. For example, the prompt line

for the outermost level of the facility is:

```
Sys Options: D(efine), I(nventory), Z(Initialize), Q(uit),
    R(emlation)
>
```

By typing "D" the user will "descend" a level within the structure diagram into a level called "DEFINE". Upon entering DEFINE, another prompt-line detailing the set of commands available at the DEFINE level of the facility is displayed. The Q(uit) command causes the user to exit from DEFINE and "ascend" back to the outermost command level of the program. Now the user is back at the level in the program from which he started. Some commands within the program prompt the user for more information, such as the name of a relation, a line number, etc. In these cases, the user enters the required information followed by a carriage return (<cr>). If an error is made in typing a portion of the information, the backspace key, CTRL H, or equivalent key depending upon the system configuration may be used to "back over" and erase the erroneous part. If the user decides not to enter any information at all, escape from most commands is by not entering any characters, ie. type <cr>. Unless otherwise stated, any input to a yes/no question besides a "Y" is considered a no.

FORMAT, KEY WORDS AND NAMES * SECTION 2

As stated previously, the DDL Facility allows the DBA to define domain and relation definitions. The format for the domain and relation definitions are predefined by the system

software. They are given as follows. A domain definition consists of the following information:

NAME = domain name

TYPE = CHAR/TEXT/INTEGER/REAL

N,M = maximum number of characters in a value of a particular domain

where N is the following:

-- the maximum number of characters if type is CHAR or TEXT

-- the maximum number of digits if type is integer

-- the maximum number of digits before the decimal point if type is real

and where M is the following:

-- the number of digits after the decimal point if the type is real.

The format for the relation definition is the following:

```
NAME       = relation name
ATTRIBUTES = list of attributes
     NAME        = attribute name
     CLASS       = classification level of the attribute
     DOMAIN      = domain name
     CONSTRAINTS = list of constraints
     SORTED UP/DOWN/NOT SORTED
```

where CONSTRAINTS has the following format:

```
OPERAND = L(ist) of values
          G(reater than)
          S(less than)
          E(qual)

VALUE or LIST of VALUES
```

It should be mentioned that when defining domain and relation definitions the user is prompted for the appropriate information. All information entered by the DBA except for the

relation name and the classification level of an attribute may be any combination of 1 to 80 nonblank characters. The relation name may be any combination of 1 to 15 alphanumeric characters. In addition, the characters ".", "-", "_", and "/" are allowed. User defined names should not be any of the following keywords:

ALL	DIVIDE	JOIN	PRODUCT	UNION
AVG	FROM	MAX	PROJECT	VETO
BY	GIVING	MIN	SELECT	WHERE
COUNT	IN	ONE	SORT	$⊖$
DIFFERENCE	INTERSECT	OVER	SUM	

The classification level of an attribute consists of a single digit, represented by a 1 for Students, a 2 for Teachers, a 3 for Secretaries, a 4 for Laboratory Technicians, and a 5 for Administrators. In addition, each higher classification level has access to all lower classification levels. That is, 1 < 2 < 3 < 4 < 5.

HOW TO USE THE DDL FACILITY * SECTION 3

3.1 Starting the System

To start the system, execute the code file DDL.CODE. The disk which contains this file must remain on-line during the execution of the system in order to permit segment swapping. However, the disk may be in any drive.

3.2 The Logon Procedure

The DDL Facility is only to be used by the DBA. This is to allow centralized control over the construction of domain and relation definitions. To insure that only the DBA has

access, a special identification name is assigned to the DBA. The indentification name is bb$$RROOTTHH. The "b" represents a blank and must be entered as part of the identification name. In order to change the identification name, the constant DBMID which is declared in the COMMON UNIT must be changed. Once the DDL Facility is completely tested, the DBA should remove the identification name from the DDL source code and put a new identification name, known only to the DBA, into a relation. This relation will be accessed whenever the DDL.CODE is executed. Then if the USERID entered matches the relation ID permission to access the DDL options will be granted, otherwise permission will be denied. If permission is denied then an error message will be written into the ERROR.DATA file for use by the DBA.

To logon to the DDL Program, a set procedure must be followed. The program indicates that it has been properly initiated by welcoming the user to the AFIT RELATIONAL DATABASE SYSTEM and displaying its version number. Next, the user is prompted for his identification number as follows:

Enter Identification -->

Here, the DBA should enter the special identification name that differentiates him from other users. If the correct id is entered, the program responds by welcoming the DBA to the system with the following prompt:

HELLO BOSS -- YOUR WISH IS MY COMMAND

Otherwise, the program indicates to the user that entrance has been denied by the following prompt,

```
ACCESS NOT ALLOWED *****
```

writes the following error message into the ERROR.DATA file,

```
ERROR => 1
ID => USERID
```

and promptly terminates.

Once the DBA has properly logged on, he is then asked if the system has 1 or 2 disk drives. If 2 drives are indicated, then the domain and relation definitions are assumed to exist on the disk drive in DRIVE 1, in a file called SETUP.DATA. Otherwise DRIVE 0 is assumed to contain the diskette with SETUP.DATA. If SETUP.DATA cannot be found, the DBA is asked if he would like one created by the following message:

```
Can't find SETUP.DATA on the disk in Drive 0/1.
Should I create one? --(Y/N)-->
```

Here, the DBA should answer with a Y or N. If the DBA indicates that he would like to create a file called SETUP.DATA, the program then writes a file called SETUP.DATA to DRIVE 1, if there are two disk drives. If only one disk drive, SETUP.DATA is written to DRIVE 0. If the DBA should decide not to create a new SETUP.DATA, the program will respond with the following prompt:

```
Insert proper disk in DRIVE 1 & hit return -->
```

The program will respond with this message until SETUP.DATA can be properly found or until the program has been halted by hitting the break key. Once SETUP.DATA is found, the Access Control Tree Setup module is executed. This module reads from the file ACLSET.DATA the relation names, ownerids, and lists of userids with associated access rights into a binary tree in memory. If ACLSET.DATA can not be found on Drive 0, the DBA is asked if he would like one created by the following message:

```
Can not find ACLSET.DATA on Drive 0
should I create one??---(Y/N)==>
```

Here, the DBA should answer with a Y or N. If the DBA indicates that he would like to create a file called ACLSET.DATA, the program then writes a file called ACLSET.DATA to DRIVE 0. If the DBA answers with a N, the DBA will be denied access to the DDL options and the program will promptly terminate. Assuming ACLSET.DATA is found, the outer level prompt line is displayed as follows:

```
Sys Options: D(efine), I(nventory), Z(Initialize),
    Q(uit), R(emrelation)
```

Each option is described in detail in the following section.

3.3 Outermost Level Commands

A. I(nventory)

To obtain a listing of all domain and relation definitions type "I" which will start the execution of the Inventory procedure. The Inventory procedure allows a list of all the domain and relation definitions currently defined to be

printed on the CRT. The format for the domain listing is the following:

List of DOMAINS defined:

NAME = domain name

TYPE = Char/Text/Integer/Real (N,M)

.
.
.

where N is the following:

-- the maximum number of characters if type char or text

-- the maximum number of digits if type is integer

-- the maximum number of digits before the decimal point if type is real

and where M is the following:

-- the maximum number of digits after the decimal point if type is real.

The format for the relation definitions is the following:

List of Relations defined:

Name = Relation Name/No relations are currently defined

Attributes are:
  Name = attribute name
  Domain = domain name
  Sorted UP/Sorted DOWN/Not sorted

Constraints = op value/No constraints

Keys are:
  Key attribute name

Existence is permanent/temporary

No. of tuples = tuple number

Stored in file = filename

To allow the user to take his time in viewing the domain and relation definitions, the user is given some control over the automatic scrolling capability. Specifically, when the screen becomes full, the continued display of the definition is halted. The following message is printed on the screen:

*** TYPE Return to continue **

If the user enters a (<cr>), display of the domain or relation definition is continued until the screen again becomes full or all the definitions have been viewed. If the screen again becomes full, the message is p inted on the CRT. This process continues until all domain and relation definitions have been displayed.

B. Z(Initialize)

To initialize the database system, the DBA must type a "Z". This destroys all domain and relation definitions. Cnce this process has been completed the following message is displayed:

Initialization Complete

C. R(emrelation)

To remove a relation name and all the information associated with this relation from the Access Control Tree the DBA must type a "R". Once this value has been entered the following prompt will be displayed:

```
Enter name of relation to be removed from Access
Control Tree =>
```

Here the DBA should enter the name of the relation to be removed as requested by the owner. If the name of the relation is not contained in the DDL the following error message will be displayed:

```
Relation Name => xxxxxx not defined in DDL
```

In addition, the following error message is written into the ERROR.DATA file,

```
ERROR => 2
ID    => USERID
RELATION NAME => relation name
NOT DEFINED IN THE DDL!!
```

and processing returns to the calling module.

D. D(efine)

To define relation or domain definitions type a "D" at the outermost level and the following prompt will be displayed:

```
1)  Define Domains
2)  Define a New Relation
3)  Input from Mass Storage
4)  Output to Mass Storage
5)  Quit

Select 1 - 5 -->
```

The individual define commands are invoked by typing the number to the left of the parenthesis.

3.4 Innermost Define Commands

A. Define Domains

To construct a domain definition enter a "1" and the user is then prompted for the domain name as follows:

```
Enter DOMAIN name --->
```

Here, the DBA should enter a combination of 1 - 80 nonblank characters. Once a domain name has been entered the following prompt is displayed:

```
Enter DOMAIN Type: C(haracter), T(ext), R(eal), I(nteger)-->
```

Here, the user should enter a "C", "T", "R", or an "I" depending on the type of domain he wishes to have defined. If an R is entered, the user is then asked to enter the maximum number of digits before the decimal point and the maximum number of digits after the decimal point as follows:

```
Enter max no. of places before decimal point-->
Enter max no. of places after decimal point -->
```

The user should respond by entering integer values. Otherwise, a zero will be assumed. If the domain type entered is other than a real number, the user is prompted as follows for the maximum number of characters or digits allowed.

```
Enter max no. of characters or digits allowed-->
```

If the domain has previously been defined, an appropriate message is displayed and the user is asked the following question:

```
Do you still want to define this domain? -(Y/N)->
```

If the answer is yes, the process of prompting the user for the domain name and type is repeated.

If the domain is properly defined, the user is next asked if he would like to define another domain. If the user desires to define another domain, the process is repeated. Otherwise, the user is no longer prompted for information and the process of defining domains is terminated.

B. Define a New Relation

Defining a new relation is a more complex process than defining a domain. To construct a relation definition, the user is first prompted for a relation name as follows:

```
Enter relation name --->
```

Here, the user should enter a combination of 1 to 15 characters. Next, the user is asked for a list of attributes and the corresponding domain name as shown:

```
Enter attribute name, return, classification level,
return and domain name for this attribute, after each
prompt (>).
Type Return alone to quit attribute definitions.
```

An example of how the user should respond is given as follows:

```
>ATTRIBUTE NAME (<cr>)
>CLASSIFICATION LEVEL (<cr>)
>DOMAIN NAME (<cr>)
>(<cr>)
```

Next, the user is asked if a directory should be maintained for the attribute and if so, whether or not it should be sorted in

ascending order or descending order. To indicate that the directory should be sorted in ascending order, the user should enter a "U". If the directory is to be sorted in descending order, the user should enter a "D". Note that specifying that a directory should be maintained is optional. However, if a directory is maintained, querying processing time may be greatly decreased. Next, the user is asked to enter the number of attributes that will serve as a key. Then the user is prompted for each attribute name that is the key value. If the attribute entered has not been defined the user is given an appropriate error message and is prompted again for the attribute name. In addition, a message is displayed if the attribute entered has already been defined as a key.

Finally, the user is asked if constraints should be made for an attribute by the following:

```
Do you want any constraints? --(Y/N)-->
Constraints on what attribute?
```

The user has the following options if constraints are desired.

```
Enter type:  E(equals), S(less than), G(reater than), or
             L for a list of values --->
```

Here, the user indicates his preference by entering either an "E", "S", "G" or "L". After an option is chosen, the program responds with the following prompt if the "L" option is not chosen:

```
Enter constraining value --->
```

Here the user should enter an appropriate value depending on the domain type. If a list of constraints is specified the system responds as shown:

```
Enter a value at each prompt.
Type RETURN alone if no more.
```

Here the user is prompted for each individual value. Values entered should adhere to domain constraints specified. Note that specifying constraints is optional. In addition, more than one constraint can be specified for an attribute. If so, all values entered for that attribute must meet at least one of the constraints specified.

Once the desired constraints have been specified, the relation is checked to determine if it has been previously defined. If so, the user is prompted for a new relation name. Otherwise the process of defining a new relation is complete.

COMMAND SUMMARY * SECTION 4

4.1 D(efine)

A. Define Domains

Define domains is used to construct a domain definition (refer to section 2). The user is prompted to enter the domain name, the type of domain, ie. Character, Integer, Text, or Real, and the number of characters or digits to be allowed. The domain name must be unique. If other than digits are entered when required then 0 is assumed; and if the maximum integer size of the machine is exceeded, then the maximum integer is used. All yes/no questions must be answered with a

"Y" or "N".

B. Define a New Relation

This command is used to define a new relation to consist of the following parts each prompted for individually:

Relation name -- must be unique and no longer than 15 characters.

Attribute name/classification level/domain name triples -- attribute names must be unique within a relation, classification level must be either a 1,2,3,4, or 5, and the domain names must be previously defined. The user is also asked if a sorted directory is to be maintained for each attribute.

Primary key(s) -- the number of keys is entered, and then each attribute which is to serve as a key is entered. The attribute must have been defined and if a duplicate is entered, the user is allowed to prematurely quit specifying keys. The primary key attributes must be classified at a level dominated by the classification levels of all the remaining attributes in the relation. In addition, all the primary key attributes must have the same classification level assigned.

Constraints -- the user is allowed to specify constraints on the attributes of this relation. These are used to further restrict the domain of an attribute. The constraints can be of the form:

$$\text{attribute} \begin{cases} > \\ < \\ = \end{cases} \text{value}$$

or

attribute = (value1, value2, value3,...,valueN).

Multiple constraints can exist on an attribute.

C. Input from Mass Storage

This command has not yet been implemented. Its purpose is to read in data from a disk file into a relation, specifying the format of the data. Data which is incompatible with the

domain type or constraints of the relation attributes should be flagged as an error, and non unique key values should also be flagged.

D. Output to Mass Storage

This command has not yet been implemented. Its purpose is to write data in a relation to disk or to a printer in a particular format.

4.2 I(nventory)

This command causes a list of the domains which have been defined and a list of the relations which have been defined and attached to be displayed on the CRT. Note that since this is the DBA routine, all relations are attached and therefore displayed.

4.3 Z(Initialize)

This command allows the DBA to initialize the system variables. This consists of returning all storage and setting the domain and relation definition lists to empty.

4.4 R(emrelation)

This command allows the DBA to remove a relatiion name and all the information pertaining to this relation from the Access Control Tree. This option will only be initiated by the DBA upon owner request.

ERROR DEFINITIONS * SECTION 5

5.1 Overview

The ERROR.TEXT file is used by the DDL.CODE file and the DATABASE.CODE file for writing error messages into the ERROR.DATA file. This file should be used only by the DBA for monitoring daily security violations.

It is strongly suggested that the DBA transfer the ERROR.DATA file to the printer so any security violations for that day can be rectified by the DBA and retained for future reference. Once the ERROR.DATA file has been transferred to the printer it should be removed from the directory and a new and empty ERROR.DATA file should be created. Again it is highly recommended that the DBA follow this sequence of procedures. However, since the ERROR.DATA file has been set up as an INTERACTIVE file, the Pascal RESET procedure will only mark the existing file open for I/O. Consequently, the file pointer points to the last record written to the ERROR.DATA file (Ref 5:121). Thus, if the DBA forgets to follow the previous procedures, the security violations for that day will not be written over or destroyed.

5.2 Errors defined

The ERROR.TEXT file defines eleven possible error conditions. Two of these error conditions may be generated by executing the DDL.CODE file and the other nine may be generated by executing the DATABASE.CODE file. A discussion of each error condition will follow.

The first two error conditions pertain to the execution of the DDL.CODE file.

```
ERROR => 1
ID    => BOSSID
```

This error condition is generated by the DB program when the BOSSID is not equal to the DBMID. This error should occur only when the DDL.CODE file is executed by someone other than the DBA.

```
ERROR => 2
ID    => BOSSID
RELATION NAME => RELNAME
NOT DEFINED IN THE DDL
```

This error condition is generated by the REMRELATION procedure when the DBA tries to remove a relation name (upon owner request) from the Access Control Tree that has not previously been defined in the DDL.

The next nine error conditions pertain to the DATABASE.CODE file which is executed by the users.

```
ERROR => 3
ID    => BOSSID
VALID LOGON
```

This error condition is generated by the DB_ACCESS_MONITOR procedure whenever the USERID and password entered by a user matches the SAVE-ID and SAVEPSW retrieved from the ACCESS relation.

```
ERROR => 4
ID    => BOSSID
INVALID LOGON ATTEMPT
```

This error condition is generated by the DB_ACCESS_MONITOR procedure whenever a USERID and password entered by a user does

not match the SAVE-ID and SAVEPSW retrieved from the ACCESS relation.

```
ERROR => 5
ID    => BOSSID
RELATION NAME => RELNAME
```

This error condition is generated by the UPDATE_ACL procedure whenever the owner of a relation tries to add the name of the relation to the Access Control Tree when that relation name has not been previously defined in the DDL.

```
ERROR => 6
ID    => BOSSID
RELATION NAME => RELNAME
```

This error condition is generated by the UPDATE_ACL procedure whenever the relation name and OWNERID entered by a user does not match the relation name and OWNERID entered in the Access Control Tree by the owner of the relation.

```
ERROR => 7
ID    => BOSSID
RELATION NAME => RELNAME
INVALID USERS NAME
```

This error condition is generated by the RETR_USERID procedure whenever a USERID is not retrieved from the ACCESS relation, that matches the USERNAME entered by the owner of a relation in which access is to be granted.

```
ERROR => 8
ID    => BOSSID
RELATION NAME => RELNAME
USER ACCESS NOT ALLOWED
```

This error condition is generated by the VALIDID procedure whenever a user tries to access a relation in which permission to access has not been granted by the owner of the relation.

```
ERROR => 9
ID    => BOSSID
RELATION NAME => RELNAME
NOT AUTHORIZED TO MODIFY
```

This error condition is generated by the GOODMODLIST function whenever a user who has access to a relation tries to modify a relation attribute which has a classification level greater than the user's classification level.

```
ERROR => 10
ID    => BOSSID
RELATION NAME => RELNAME
UNAUTHORIZED TO DELETE
```

This error condition is generated by the GOODSELLIST function whenever a user tries to delete a tuple from a relation in which the user has not been granted delete access.

```
ERROR => 11
ID    => BOSSID
RELATION NAME => RELNAME
UNAUTHORIZED TO INSERT
```

This error condition is generated by the GOODINPUT function whenever a user who has access to a relation tries to insert values into relation attributes which have a classification level greater than the user's classification level.

CHANGING THE SYSTEM * SECTION 6

6.1 Overview

Please refer to sections 3.3.1 and 3.3.2 of the UCSD Pascal Version II.0 reference manual and current program listings before trying to change the system. The program is currently divided into a main body segment, two segment procedures and a unit. Each segment is contained in a dummy program in order to permit seperate compilation. The unit contains all types, variables, and procedures which are global to more than one segment. Thus by including the unit in each dummy program, access is allowed to those elements. The format for each segment procedure is:

```
PROGRAM dummy-name;
USES COMMON;  (*the unit is named COMMON*)
SEGMENT PROCEDURE name(parameter-list);
  Local types, variable, and procedures;
BEGIN
  body of name;
END; (*name*)
BEGIN
END. (*dummy name*)
```

Since the segments are separately compiled, the parameter list of the segment procedure must contain all global variables accessed or modified by the procedure. The program or segment procedure which calls each segment procedure must have a dummy segment procedure with the same name, so that it may compile properly. The format for the main body segment is:

```
PROGRAM main;
USES COMMON;
 local labels,types,constants, and variables;
SEGMENT PROCEDURE name1( --- );
BEGIN
END; (*name1*)
SEGMENT PROCEDURE name2( --- );
BEGIN
END; (*name2*)
```

```
.
.
.
other local procedures;
BEGIN
  body of main;
END. (*main*)
```

The format for segment procedures which use other segments is the same as the previous format for a segment procedure except dummy segment procedures are included as local procedures.

Each segment procedure has a particular segment number from 11 to 15 associated with it. The main body segment has number 1 and the unit has number 10. Other numbers are for Pascal use only. The way numbers are assigned to the segment procedures is in first compiled, first numbered order. Thus, in the above format for the main body segment name1 would be assigned 11, name2 assigned 12, etc. Therefore in each dummy program used to define a segment procedure, an appropriate number of dummy segments must exist before the defined segment to ensure that the segment count is the same. Thus, for example, the format for segment name2 would be:

```
PROGRAM dummy name;
USES COMMON;
SEGMENT PROCEDURE dummy name1;
BEGIN
END; (* dummy name1 *)
SEGMENT PROCEDURE name2( --- );
  local labels, types, etc.
BEGIN
  body of name2
END; (*name2*)
BEGIN
END.  (*dummy name*)
```

The unit -- COMMON -- is compiled and placed in the system library using the librarian program, LIBRARY.CODE. (See section 4.2 of the UCSD Pascal manual.) When each program is compiled, the unit is retrieved from the system library and used in the program. However, each program must still be linked with the system library in order to bind the external variable and procedure references into the unit. After each program is compiled and linked, then the librarian may be used to put all the segments together into one code file. Each code file containing the segment is retrieved and linked into the proper space using its assigned segment number into the overall file.

If a particular segment, including the main body segment, is to be changed then the steps to be followed are:

1) Change the source code for the segment.

2) Compile the program containing the segment.

3) Link the code file to the system library.

4) Using the librarian create a new overall file passing all unchanged segments to the new file and linking in the changed segment.

If the unit has to be changed then after compiling it and placing it into the system library, each program must be recompiled, linked to the system library, and then put together with the librarian.

6.2 Bringing Up the Data Definition Facility

To bring up the Data Definition Facility the following steps should be followed:

1. First, compile COMMON.TEXT such that the object code resides in COMMON.CODE.

2. Execute the System Librarian by typing "X" at the system level. The following prompt line will appear on the CRT:

   EXECUTE WHAT CODE FILE --->

3. Enter "LIBRARY".

4. Next, the following prompt will appear:

   OUTPUT CODE FILE --->

5. Enter a "*" or "SYSTEM.LIBRARY".

6. Next, the user will be prompted as follows:

   LINK CODE FILE --->

7. Enter #COMMON.CODE where # is #4/#5 and the screen will appear as follows:

```
LINK CODE FILE ---> #COMMON.CODE

0-        0  4-        0  8-        0 12-        0
1-        0  5-        0  9-        0 13-        0
2-        0  6-        0 10-COMMON     14-        0
3-        0  7-        0 11-        0 15-        0

OUTPUT CODE FILE ---> *
```

The user may now link the COMMON segment into segment 10 of the output code file by typing a "10", (<cr>) and "10". In addition to linking this segment into the SYSTEM.LIBRARY, which resides on the system disk, the two segments PASCALIO and DECOPS must also be linked back into the System library, SYSTEM.LIBRARY. In

order to do this, type "N" for new. This indicates to the librarian that segments from a new code file are to be put into the output code file. The librarian will respond with the following prompt:

```
LINK CODE FILE --->
```

8. Enter a "*" or "SYSTEM.LIBRARY".

9. To link PASCALIO, type a "2", followed by a (<cr>), then a "2".

10. To link DECOPS, type a "3", followed by a (<cr>), then a "3".

Now, the SYSTEM.LIBRARY should look like the following:

```
0-              0 4-       0  8-          0 12-        0
1-              0 5-       0  9-          0 13-        0
2-PASCLIO    1824 6-       0 10-COMMON      14-        0
3-DECOPS     2092 7-       0 11-          - 15-        0
```

To exit from the Librarian do the following:

11. Type "Q" for QUIT.

12. ENTER (<cr>) to exit LIBRARIAN.

At this point, the DDL software can be compiled. This software resides in the following text files:

DBMGR.TEXT - the executive for the DDL Facility.

DBDUM1.TEXT - the file that contains the source code for the Define segment.

DBDUM2.TEXT - the file that contains the source code for the Inventory segment.

QUIT.TEXT - the file that contains the source code for writing domain and relation definitions to disk.

SETREL.TEXT - the file that contains the source code for reading from disk (ACLSET.DATA) the relation names, ownerids, and list of userids with associated access rights into a binary tree in memory.

REMREL.TEXT - the file that contains the source code for removing a relation name and all the information pertaining to this relation from the Access Control Tree.

ERROR.TEXT - the file that contains the source code for writing to disk (ERROR.DATA) any logon or illegal transactions.

To link the DDL source code together, continue the following steps:

13. Compile DBMGR.TEXT making sure that QUIT.TEXT, SETREL.TEXT, REMREL.TEXT, and ERROR.TEXT reside on the same disk. Next, compile DBDUM1.TEXT, and DBDUM2.TEXT.

14. Once all three source files have been compiled properly into three appropriate CODE files, they can be linked together into one CODE file using the LIBRARIAN. To invoke the LIBRARIAN, repeat steps 2 and 3. Once, the prompt has been displayed, enter a valid CODE file name.

15. Next, the following prompt will be displayed:

    LINK CODE FILE --->

16. Enter code file for DBDUM1. To link the Define segment into the output code file link segment 11 to segment 11. Next, enter a "N" for new. This will indicate to the LIBRARIAN the fact that segments from a new code file are to be linked into the output file. For the new code file enter the code file for DBDUM2 To link the Inventory segment into the output code file link segment 12 to segment 12. The final code file to be linked is the code file for DBMGR. Type "N and enter the code file name for the DBMGR object code. to link the DB segment into the output code file, link segment 1 to segment 1.

17. To finish the linking process, PASCALIO, DECOPS,

and COMMON must all be linked into this output code file. To do this type "N" and for the LINK CODE FILE enter "*" or SYSTEM.LIBRARY. To link these segments into the output code file, link segment 2 to segment 2, segment 3 to segment 3, and segment 10 to segment 10. The output code file should look like the following:

```
0-              0 4-    0  8-           0 12-Inventory
1-DB              5-    0  9-           0 13-              0
2-PASCALIO   1824 6-    0 10-COMMON       14-              0
3-DECOPS     2092 7-    0 11-DEFINE       15-              0
```

18. The final step is to link the output code file by executing the LINKER. To do so, type an "L" at the Pascal system level. The following prompts will be displayed:

```
Host file?   enter output code file that contains all
             segments to be linked.
Lib file?    enter (<cr>).
Map file?    enter any valid file name.
Output file? enter DDL.CODE, this will be the executable
             code file.
```

To execute DDL.CODE see section 3.1.

AFIT/GCS/EE/82-4

USERS'S GUIDE FOR THE ROTH RELATIONAL DATABASE SYSTEM

APPENDIX B

TO

THE ANALYSIS AND DESIGN
OF A
COMPUTER SECURITY/RECOVERY SYSTEM
FOR A
RELATIONAL DATABASE MANAGEMENT SYSTEM
THESIS

BY

WILLIAM P. REED
CAPT USAF

Graduate Information Systems
17 December 1982

CONTENTS

Page

I. Introduction and Overview. 148

The Database System: An Overview 148

II. Format, Key Words, and Names 150

III. How to Use the AFIT Relational Database System 151

Starting the System. 151
The Logon Procedure. 151

IV. Edit Procedures. 154

Insert 155
Delete 158
Modify 162
Copy 167
Transfer 167

V. Retrieve Procedures. 168

The Query Commands 168

Union of Two Relations 168
Intersection of Two Relations. 169
Difference or Relative Complement of Two Relations . . 169
Cartesian Product of Two Relations 169
Join of Two Relations. 169
Project a Relation over a Subset of Its Attributes . . 170
Select a Subset of Tuples from a Relation. 170
Divide a Binary Relation by a Unary Relation 171

Get. 175
Save 175
Edit 175

Insert 176
Delete 176
Begin. 176
Page 176
Execute. 176
Display. 177

VI. UPDATE PROCEDURES 177

VII. Command Summary. 181

Edit 181
Retrieve 181
Inventory. 182
Attach 182

Quit 182
Update 182

VIII. Changing the System. 183

APPENDIX B

USER'S GUIDE FOR
THE ROTH RELATIONAL DATABASE SYSTEM

INTRODUCTION AND OVERVIEW * SECTION 1

The database system described in this document is a system intended to run on the LSI-11 under the control of the UCSD Pascal operating system, Version II.0. All system software is written in Pascal, which should aid in the maintenance and modification of the system software. The system is designed to be used with a CRT terminal for display purposes.

1.1 The Database System: An Overview

The structure of the system is best conceptualized in terms of the tree-like structure in Figure B-1.

While a user is in a particular level, the system displays a list of available commands called the "prompt-line" at the top of the screen. Commands are usually invoked by typing a single character from the keyboard. For example, the prompt line for the outmost level of the system is:

```
Sys Options: E(dit), R(etrieve), I(nventory), A(ttach), Q(uit)
             U(pdate Access Control List)
>
```

By typing "R" the user will "descend" a level within the structure diagram into a level called "RETRIEVE". Upon entering RETRIEVE, another prompt-line detailing the set of commands available at the RETRIEVE level of the system is

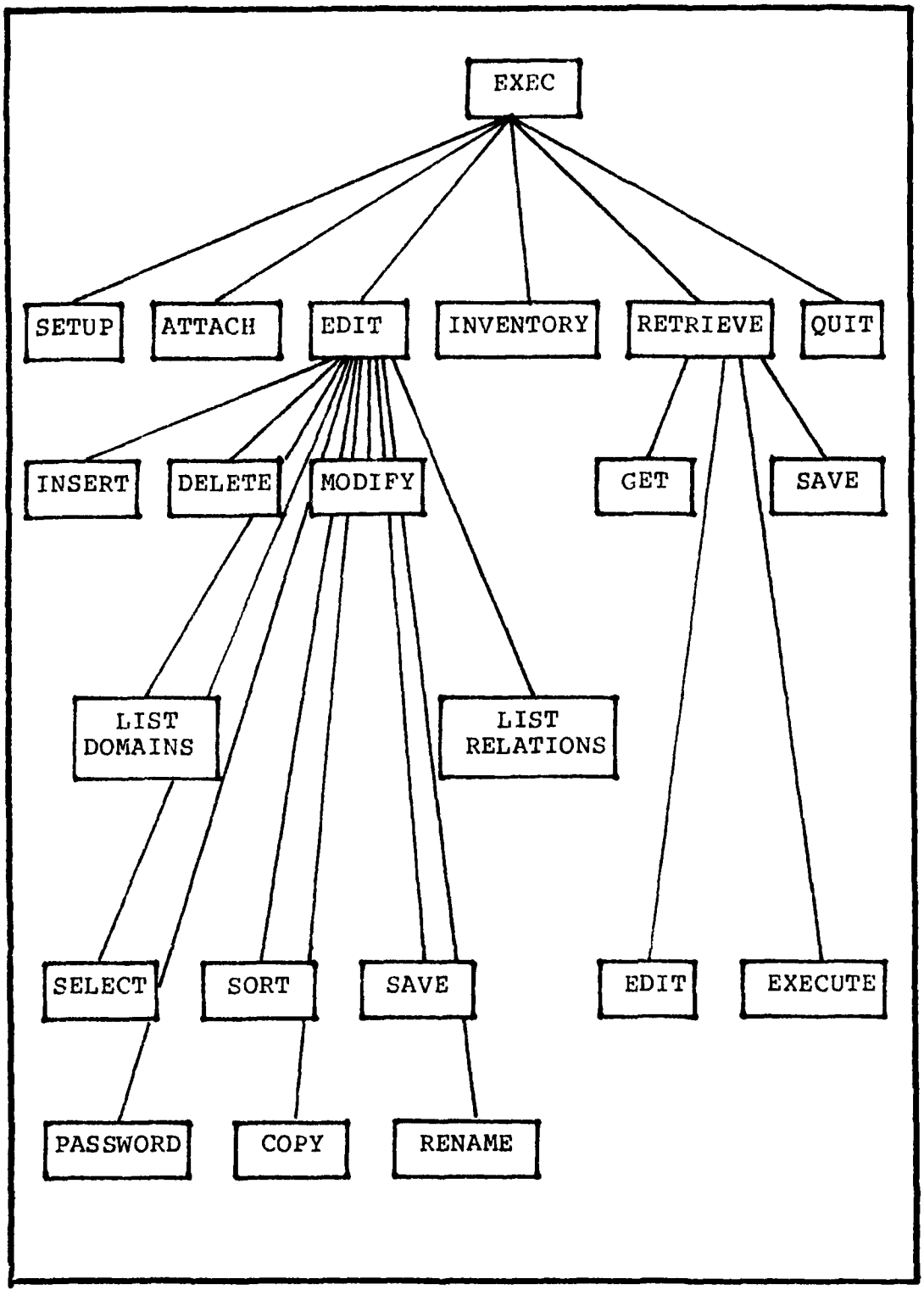


Figure B-1: Roth's System without the DDL

displayed. The Q(uit) command causes the user to exit from

RETRIEVE and "ascend" back to the outmost command level of the system. Now the user is back at the level in the system from which he started after initially executing the system. Here the user can either choose another option or he can choose to exit from the system by entering a "Q" for quit.

FORMAT, KEYWORDS AND NAMES * SECTION 2

In order to use the database system, it is a good idea to become familiar with the structure of relation and domain definitions. The format for the domain and relation definitions are predefined by the system software. They are given as follows. A domain definition consists of the following information:

NAME = domain name

TYPE = CHAR/TEXT/INTEGER/REAL

N, M = maximum number of characters in a value of a particular domain

where N is the following:

-- the maximum number of characters if type is CHAR or TEXT

-- the maximum number of digits if type is Integer

-- the maximum number of digits before the decimal point if the domain type is REAL

and where M is the following:

-- the number of digits after the decimal point if the type is real.

The format for the relation definition is the following:

NAME = relation name
ATTRIBUTES = list of attributes

```
NAME        = attribute name
CLASS       = classification level of the attribute
DOMAIN      = domain name
CONSTRAINTS = list of constraints
SORTED UP/DOWN/NOT SORTED
```

where CONSTRAINTS has the following format:

```
OPERAND = L(IST OF VALUES)
          G(REATER THAN)
          S(LESS THAN)
          E(QUAL)

VALUE OR LIST OF VALUES
```

It should be mentioned that various means of entering information into the database are used. For example, some information the user is prompted for and some information, such as a query the user has to enter in a prescribed format. Although this is the case, the user should pay close attention to the information entered. Specifically, the user should avoid using the following keywords except where specified in the syntax of a query comand:

ALL	DIVIDE	JOIN	PRODUCT	UNION
AVG	FROM	MAX	PROJECT	VETO
BY	GIVING	MIN	SELECT	WHERE
COUNT	IN	ONE	SORT	$@$
DIFFERENCE	INTERSECT	OVER	SUM	

HOW TO USE THE AFIT RELATIONAL DATABASE SYSTEM * SECTION 3

3.1 Starting the System

The system is started by executing the code file DATABASE.CODE which can be located in either drive. The disk which contains this file must remain on line during the execution of the system in order to permit segment swapping.

3.2 The LOGON Procedure

Upon execution of the system, a welcome message is displayed along with the Version number of the system. Next, the user is asked to enter his identification name by the following prompt:

Enter Identification --->

Here the user enters his nine digit, DBA assigned identification name. This name is associated with all the relations this user is allowed to access. Then the user is asked to enter his DBA assigned password by the following prompt:

Enter Password ==>

Here the user enters a nine, nonblank character password. This password is composed of the digits 0-9 and/or the characters A-Z and is associated with the identification name previously entered. Thus both the identification name and password are used as part of the security system. If the identification name and password match what is contained in the ACCESS RELATION tuple then the user is granted permission to use the DBMS by the following prompt:

ACCESS HAS BEEN GRANTED

If they do not match, the following prompt is displayed:

ACCESS HAS BEEN DENIED

In addition, the user is prevented from any further access to the DBMS and the following prompt is displayed:

```
LEAVING DATABASE
```

Regardless of whether the identification name and password match, the logon attempt will be logged into the file ERROR.DATA. If the logon attempt is successful the following information is written into the ERROR.DATA file,

```
ERROR => 4
ID => Identification name
      VALID LOGON
```

If the logon attempt is unsuccessful the following information is written into the ERROR.DATA file,

```
ERROR => 3
ID    => Identification name
         INVALID LOGON ATTEMPT
```

This information is to be used by the DBA for monitoring usage and illegal logon attempts.

The user is then asked by the following prompt whether or not his system has 1 or 2 disk drives.

```
DO YOU HAVE 2 OR MORE DISKS --(Y/N)-->
```

If 2 drives are indicated, then the domain and relation definitions are assumed to exist on the disk drive in DRIVE 1 in a file called SETUP.DATA. Otherwise, DRIVE 0 is assumed to contain the disk with SETUP.DATA. If SETUP.DATA cannot be found, the system will respond with the following prompt:

```
Insert proper disk in DRIVE 1 and hit return -->
```

The system will respond with this message until SETUP.DATA can be properly found or until the program has been halted. Once SETUP.DATA is found, the Access Control List setup module is executed. This module reads from the file ACLSET.DATA the relation names, owner-ids, and lists of user-ids with associated access rights into a tree in memory. If ACLSET.DATA can not be found on Drive 0 then the system will respond with the following prompt:

```
Can not find ACLSET.DATA on Drive 0
Insert proper disk on Drive 0 and hit return ==>
```

The system will respond with this message until ACLSET.DATA can be properly found or until the program has been halted. Once ACLSET.DATA has been found, the outer level prompt line is displayed as follows:

```
Sys Options: E(dit), R(etrieve), I(nventory), A(ttach),
             Q(uit), U(pdate Access Control List)
>
```

Each option is described in detail in the following sections.

EDIT PROCEDURES * SECTION 4

To perform any edit functions, type "E" at the system level to "descend" a level into the system editor. The following options are displayed:

```
Edit Options: I(nsert), D(elete), M(odify), C(opy), T(ransfer),
              Q(uit)
>
```

The individual Edit functions are invoked by typing the letter found to the left of the parenthesis. All Edit commands except the Copy and Transfer commands have been implemented.

4.1 I(nsert)

To insert a tuple into a relation type an "I" at the Edit level. The system immediately responds with a prompt for the relation name

```
ENTER RELATION NAME --->
```

The user should enter any combination of 1 - 15 nonblank alphanumeric characters or the following special characters: ".", "-", "_", or "/". If any other type of character is entered an appropriate error message is generated and the user is asked if he would like to correct his input. If not, the insertion process is terminated. Otherwise, the user is again prompted for the relation name. Once a relation name has been entered that is syntactically correct, it is checked as to whether or not the relation is defined. If not properly defined, the user is again asked if he would like to continue the insertion process. If not, the process is terminated. Otherwise, the user is again prompted for a relation name.

If the relation is properly defined, the relation is checked to see if it is attached. If not the user is given the option of attaching the relation. If the relation is attached, then the identification name that the user logged on with is compared to the Access Control List of user-ids associated with

the relation name previouly entered. If the identification name does not match any of the user-ids associated with this relation the following prompt is displayed,

```
You do not have access to relation => Relation Name
Please check with your DBA or relation owner!
```

the following error message is written into the ERROR.DATA file which is used by the DBA for monitoring illegal accesses to the relations,

```
ERROR => 8
ID    => USER-ID
RELATION NAME => Relation Name
USER ACCESS NOT ALLOWED
```

and processing returns to the Edit Option menu. However, if a user-id is found then the access rights associated with this user are retrieved and the user is prompted for values for each attribute in the relation. Now the user must enter a value for each attribute terminated by the line terminator which is a semicolon. The value entered must satisfy the domain constraints and value constraints specified for that attribute when it was defined. If not, the user is given an appropriate error message and asked if he would like to correct his input. If he chooses to correct his input, the screen is cleared and the attribute name is then displayed as a prompt. Otherwise, the insertion process is terminated. In addition, the classification level of each attribute is compared to the user's classification level which was retrieved at the time of logon. If the classification level of the attribute is greater

than the user's classification level the following prompt is displayed,

```
Permission to Insert denied!!!!
Please seek DBA assistance
```

the following error message is written into the file ERROR.DATA,

```
ERROR => 11
ID    => USER-ID
RELATION NAME => Relation name
NOT AUTHORIZED TO INSERT
```

and processing continues by checking the next attribute value to be inserted into the tuple. Otherwise the user is allowed to insert the attribute value into the tuple.

Once all the attributes this user is allowed to access are given valid values, the user is asked if he would like to insert more tuples. If more tuples are to be inserted, the user is again prompted for values for each attribute and the attribute classification level is verified. If no more tuples are to be inserted, the entire command is displayed. The user is then asked by the following prompt whether or not the information displayed is correct:

```
IS THIS THE CORRECT COMMAND --(Y/y or N/n)-->?
```

If the user indicates that the information displayed is indeed correct, the insertion process begins. Otherwise, it is terminated. Note, all answers to yes or no questions can be answered with either a "Y" or "y" or "N" or "n".

4.2 D(elete)

To delete a tuple from a relation type "D" at the Edit level. The system will respond with a prompt for the relation name

ENTER RELATION NAME --->

Here, the user can enter any combination of 1 - 15 nonblank alphabetic, numeric characters or the following special characters: ".", "-", "_", "/". If any other type of character is entered, an appropriate error message is generated and the user is asked if he would like to correct his input. If yes, the user is again prompted for a relation name. Otherwise, the deletion process is terminated.

Once a relation name has been entered that is syntactically correct, it is then checked as to whether or not the relation is defined. If not properly defined, an error message is printed and the user is asked if he would like to correct his input. If so, the user is again prompted for a relation name. If the user enters a relation name that has been defined, the relation is checked to see if it has been attached. If not, a message is displayed indicating that the relation is not attached. Then the user is prompted as follows:

Attach Relation Name --(Y/y or N/n)--> ?

If the user indicates with entering a "Y" or "y" that he would like to attach the relation, the relation is then attached and

the following message appears at the bottom of the screen:

ATTACHED RELATION NAME

Otherwise, if the user enters a "N" or "n", the relation is not attached and a message indicating this fact is printed at the bottom of the screen. If other than a "Y", "y", "N", or "n" is entered, the user is prompted again.

Once it has been determined that the relation is attached, then the identification name that the user logged on with is compared to the Access Control List of user-ids associated with the relation name previously entered. If the identification name does not match any of the user-ids associated with this relation, then the following prompt is displayed,

```
You do not have access to relation ==> Relation Name
Please check with the DBA or relation owner!
```

In addition, the following error message is written into the ERROR.DATA file,

```
ERROR => 8
ID    => USER-ID
RELATION NAME => Relation Name
USER ACCESS NOT ALLOWED
```

and processing returns to the Edit Option menu. However, if a valid user-id is found then the access rights associated with this user are retrieved and the user is prompted for the values for each attribute. This is done by displaying the attribute name and a prompt as follows:

ATTRIBUTE NAME
>

The user has the option of entering a value for an attribute. If he chooses not to enter a value for an attribute, (<cr>) should be entered. However, if a value should be entered, the following format should be followed:

Operator Value Connector...;

where the Operator must be one of the following:

"=" - Equal
">" - Greater than
"<" - Less than
"<>" - Not equals

and a Connector can be either "AND" or "OR". Note, a connector is optional. In addition, it should be pointed out that the blank between the operator and the value is also optional. All input may be several lines in length. Therefore, to indicate the end of input, the user must enter a semicolon. Next the input is checked to see if it is syntactically correct and if the value specified adheres to domain restrictions and constraints specified during the time the relation was defined. If there is an error in the input, an appropriate error message is displayed at the bottom of the screen. The user is then asked if he would like to correct the input. If the user chooses to correct the input the screen is cleared and the attribute name and prompt is displayed again. Otherwise, the deletion process is aborted. In addition, the classification

level of each attribute is compared to the user's classification level which was retrieved at the time of logon. If the classification level of any attribute is greater than the user's classification level the following prompt is displayed,

```
Permission to delete denied!
Please seek DBA assistance
```

the following error message is written into file ERROR.DATA,

```
ERROR => 10
ID    => USER-ID
RELATION NAME => Relation Name
NOT AUTHORIZED TO DELETE
```

and processing returns to the Edit Option menu. Otherwise the user is allowed to delete the tuple within the previously specified relation.

Once all the input is entered and the attribute classification levels are verified the deletion command is displayed in its entirety. The user is then asked to verify the deletion operation to be performed by the following question:

```
IS THIS THE CORRECT COMMAND --(Y/y or N/n)-->?
```

Once the deletion command is verified, the operation is performed. When it is properly completed, a message is displayed indicating the fact. If the deletion command is not verified, the deletion process is terminated.

Note, all yes/no questions must be answered with either a

"Y", "y", "N", or "n".

4.3 M(odify)

To modify tuples of a relation, type "M" at the edit level. The system will immediately respond with the following prompt for the name of the relation to be modified:

```
ENTER RELATION NAME --->
```

Here, the user should enter any combination of 1 - 15 nonblank alphabetic, numeric characters or the following special characters: ".", "-", "_", "/". If any other type of character is entered an error message is generated indicating this fact and the user is asked if he would like to correct his input by the following question:

```
WOULD YOU LIKE TO CORRECT YOUR INPUT --(Y/y or N/n)-->
```

If the user would like to correct the input, he should indicate this fact by entering either "Y" or "y". Otherwise, he should enter "N" or "n" and the modification process is terminated. If any other characters are entered, a message indicating that the user must enter either a "Y", "y", "N", or "n" is printed at the bottom of the screen. Also, the user is again asked if he would like to correct his input.

Once a relation name has been entered that is syntactically correct, it is checked as to whether or not the relation is defined. If not properly defined, the user is again asked if he would like to correct his input as previously described. If not, the process is terminated. Otherwise, the

user is again prompted for a relation name.

If the relation is properly defined, the relation is checked to see if it is attached. If not, a message is displayed indicating that the relation is not attached. Then the user is prompted as follows:

Attach Relation Name --(Y/y or N/n)--> ?

If the user indicates by entering "Y" or "y" that he would like to attach the relation, the relation is then attached and the following message appears at the bottom of the screen:

ATTACHED RELATION NAME

Otherwise, if the user enters a "N" or "n", the relation is not attached and a message indicating this fact is printed at the bottom of the screen. If other than a "Y/y" or "N/n" is entered, the user is prompted again.

Once it has been determined that the relation is attached, then the identification name that the user logged on with is compared to the Access Control List of user-ids associated with the relation name previously entered. If the identification name does not match any of the user-ids associated with this relation then the following prompt is displayed,

You do not have access to relation ==> Relation Name
Please check with the DBA or relation owner!

In addition, the following error message is written into the

ERROR.DATA file,

```
ERROR => 8
ID    => USER-ID
RELATION NAME => Relation Name
USER ACCESS NOT ALLOWED
```

and processing returns to the Edit Option menu. However, if a valid user-id is found then the access rights associated with this user are retrieved and the user is prompted for the criteria each attribute should meet to determine whether it should be modified. This is done by displaying the attribute name and a prompt as follows:

```
ATTRIBUTE NAME
>
```

Here, the user has the option of entering selection criteria for an attribute. If he chooses not to enter any criteria, a (<cr>) should be entered. However, if selection criteria should be entered, the following format should be followed:

```
Operator Value Connector...;
```

where the Operator must be one of the following:

```
"=" - Equal
">" - Greater than
"<" - Less than
"<>" - Not equal
```

and a Connector can either be "AND" or "OR". Some examples of syntactically correct entries are shown as follows:

```
RANK
> =2LT OR = CAPT;
```

```
AGE
> <38;
CURRENTPCS
> = Wright Patterson OR =Eglin;
```

Note, that the blank between the operator and the value is optional. In addition, all input can be several lines in length. Therefore, to indicate the end of input, the user must enter a semicolon.

Next, the input is checked to see if it is syntactically correct and if the value specified adheres to domain restrictions and constraints specified. If there is an error in the input, an appropriate error message is displayed at the bottom of the screen. The user is then asked if he would like to correct the input. If the user chooses to correct the input, the screen is cleared and the attibute name and prompt is displayed again. Otherwise, the deletion process is terminated.

Once all selection criteria is entered, the user is prompted for the values for the attributes which are to be modified. This is done by displaying each attribute name and a prompt as follows:

ATTRIBUTE NAME=

Here, the user has the option of entering a value. If he chooses not to enter a value for the attribute, the user should enter a (<cr>). Otherwise, if a value is entered, it must adhere to the domain restrictions and constraints specifed for that attribute. If it does not meet these restrictions, an

appropriate error message is displayed and the user is asked if he would like to correct his input. If not, the modification process is terminated. Otherwise, the screen is cleared and the attribute name is then displayed again. In addition, the classification level of each attribute is compared to the user's classification level which was retrieved at the time of logon. If the classification level of any attribute is greater than the user's classification level the following prompt is displayed,

```
Permission to modify denied!!!
Please seek DBA assistance
```

the following error message is written into the ERROR.DATA file,

```
ERROR => 9
ID    => USER-ID
RELATION NAME => Relation Name
NOT AUTHORIZED TO MODIFY
```

and processing returns to the Edit Option menu. Otherwise the user is allowed to modify the attribute value of a tuple within the relation.

Once all the desired values are entered and the attribute classification levels are verified, the modification command is displayed in its entirety. The user is then asked if this is the command he wishes to execute as follows:

```
IS THIS THE CORRECT COMMAND --(Y/y or N/n)-->?
```

If the user indicates that the information displayed is

correct, the modification process is carried out. Otherwise, it is terminated. Note, all answers to yes/no questions must be answered with either a "Y", "y", "N", or "n".

4.4 C(opy)

This command has not yet been implemented. However, its function is as follows. The Copy command is used to copy a tuple from one relation to another. Both relations should be previously defined and attached. The user must have been granted read access to the source relation and insert access to the destination relation. In addition the identification name that the user logged on with will have to be compared to the Access Control List of user-ids associated with the two relations. It will also be necessary to verify the attribute classification levels to the user's classification level for read and insert.

4.5 T(ransfer)

This command has not yet been implemented. However, its suggested function is described as follows. The Transfer command is used to move a tuple from one relation, the source, to another previously defined relation, the destination. When the tuple has been moved to the destination relation, it is then removed from the source relation. In order to complete this transaction it will be necessary for the user to have been granted delete access to the source relation and insert access to the destination relation. In addition the identification name that the user logged on with will have to be compared to

the Access Control List of user-ids associated with the two relations. It will also be necessary to verify the attribute classification levels to the user's classification level for insert and delete.

RETRIEVE PROCEDURES * SECTION 5

Type "R" at the system level to enter RETRIEVE and the following prompt line is displayed:

```
Retrieve ops: G(et), S(ave), E(dit), X(ecute), D(isplay), Q(uit)
>
```

A concept central to the operation of RETRIEVE is the command file. A command file contains one or more relational queries. The command file can be created, modified, stored on disk, retrieved from disk, and executed. The commands which can reside in a command file are described below. Several examples are provided after that.

5.1 THE QUERY COMMANDS

A. Union of two relations:

```
UNION relation1, relation2 GIVING relation3
```

where the first two relations must be union-compatible; that is, they have the same number of attributes and the ith attribute of one relation must be drawn from the same domain as the ith attribute of the other relation. Relation3 will acquire the attribute names of relation1. Relations 1 and 2 must have been attached, the user-id must be verified through the Access Control List, attribute classification levels must

be verified using the user's classification level, and relation3 must be unique.

B. Intersection of two relations:

INTERSECT relation1, relation2 GIVING relation3

where all restrictions under UNION apply.

C. Difference or relative complement of two relations:

DIFFERENCE relation1, relation2 GIVING relation3

where relation3 = relation1 - relation2. All restrictions under UNION apply.

D. Cartesian product of two relations:

PRODUCT relation1, relation2 GIVING relation3

where attribute names in relation3 will be the same as those in relation 1 and 2 except that duplicate names will be prefixed by the name of the relation it came from. Relations 1 and 2 must have been attached, the user-id must be verified through the Access Control List, attribute classification levels must be verified using the user's classification level, and relation3 must be unique.

E. Join of two relations:

JOIN relation1, relation2 WHERE attr1 op attr2 GIVING relation3

where attr1 is in relation1 and attr2 is in relation2, op

is =, <, >. The JOIN operation is a subset of the cartesian product where the condition of membership is specified in the WHERE clause. All restrictions under PRODUCT apply.

F. Project a relation over a subset of its attributes:

PROJECT relation1 OVER attr1,attr2,...,attrN GIVING relation2

where attributes not specified in the OVER clause will be eliminated and any duplicate tuples will be eliminated. Relation1 must have been attached, the user-id must be verified through the Access Control List, attribute classification levels must be verified using the user's classification level, and relation2 must be unique.

G. Select a subset of tuples from a relation:

SELECT ALL FROM relation1 WHERE condition GIVING relation2

where condition is a boolean predicate on the attributes of relation1 of the form a1 AND/OR a2 AND/OR a3 ... , where each aN is of the form attribute op value, where op is =, <, or >. The expression may be fully parenthesised to indicated the proper precedence of the operators, but if not then AND has precedence over OR. One or more blanks or commas must be between each part of the command except that the left parenthesis may be flush against an item to its right, and the right parenthesis may be flush against an item to its left. Relation1 must have been attached, the user-id must be verified through the Access Control List, attribute classification

levels must be verified using the user's classification level, and relation2 must be unique.

H. Divide a binary relation by a unary relation:

DIVIDE relation1 BY relation2 OVER attr1 GIVING relation3

where relation1 is a binary relation, relation2 is a unary relation, and attr1 is an attribute of relation1 defined on the same domain as the attribute in relation2. Relation3 will be a unary relation with attribute from relation1 not attr1. Relation 1 and 2 must have been attached, the user-id must be verified through the Access Control List, attribute classification levels must be verified using the user's classification level, and relation3 must be unique.

Examples: The following relations are used as the basis for examples:

Relation: part, Key: part#

part#	name	location	color
56	wheel	Miami	silver
78	cam	Boise	red
79	tire	Boise	black
100	seat	Dayton	green
711	fender	Miami	black
899	clock	Enon	red
1245	light	Boise	silver

Relation: shipment, Key:(part#,supply#)

part#	supply#	quantity
78	4567	890
100	45	900
100	546	50
100	4567	435
899	2309	1000
899	4567	13

Relation: shippart, Key: part#

part#	name	location	color
78	cam	Boise	red
100	seat	Dayton	green

899	clock	Enon	red
1000	cap	Dayton	blue

Using these relations examples of the output of the above commands are shown beneath the command.

UNION shippart, part GIVING upart

upart

part#	name	location	color
56	wheel	Miami	silver
78	cam	Boise	red
79	tire	Boise	black
100	seat	Dayton	green
711	fender	Miami	black
899	clock	Enon	red
1000	cap	Dayton	blue
1245	light	Boise	silver

INTERSECT shippart, part GIVING ipart

ipart

part#	name	location	color
78	cam	Boise	red
100	seat	Dayton	green
899	clock	Enon	red

DIFFERENCE shippart, part GIVING dpart

dpart

part#	name	location	color
1000	cap	Dayton	blue

PRODUCT shipment, dpart GIVING ppart

ppart

shipment-part#	supply#	quantity	dpart-part#	name	location	color
78	4567	890	1000	cap	Dayton	blue
100	45	900	1000	cap	Dayton	blue

```
100          546        50        1000        cap   Dayton    blue
100         4567       435        1000        cap   Dayton    blue
899         2309      1000        1000        cap   Dayton    blue
899         4567        13        1000        cap   Dayton    blue
```

```
JOIN part, shipment WHERE part# = part# GIVING shipment-
description
```

```
shipment-description

part
-part#  name   location   color   shipment-part#   supply#   quantity

78      cam    Boise     red           78           4567       890
100     seat   Dayton    green        100             45       900
100     seat   Dayton    green        100            546        50
100     seat   Dayton    green        100           4567       435
899     clock  Enon      red          899           2309      1000
899     clock  Enon      red          899           4567        13
```

```
PROJECT shipment-description OVER color, supply# GIVING c-and-s
```

```
c-and-s

color   supply#

green    45
green    546
green    4567
red      4567
red      2309
```

```
SELECT ALL FROM part WHERE location = Miami GIVING parts-
in-Miami
```

```
parts-in-Miami

part#   name   location   color

56      wheel  Miami      silver
711     fender Miami      black
```

```
SELECT ALL FROM shipment WHERE (supply# > 4000 AND quantity
< 800) GIVING s-q
```

```
s-q

part#   supply#   quantity

100     4567      435
```

```
899     4567      13

PROJECT shipment OVER part#, supply# GIVING ps#

ps#

part#  supply#

78       4567
100        45
100       546
100      4567
899      2309
899      4567

PROJECT s-q OVER supply# GIVING s#

s#

supply#

4567

DIVIDE ps# BY s# OVER supply# GIVING p#

p#

part#

78
100
899
```

Each RETRIEVE command may be split between two or more lines in the command file if the split is made at the key word. For example, some of the ways the last command in the examples above could be split is as follows:

```
DIVIDE ps#     DIVIDE ps# BY s#  DIVIDE ps# BY s# OVER supply#
by S#          over supply#      GIVING p#
OVER supply#   GIVING p#
GIVING p#
```

The commands may be combined in any sequence to formulate

one or more queries. For example, the query "Find the colors of all parts supplied by any supplier in quantity > 500" can be expressed as:

```
SELECT ALL FROM shipment WHERE quantity > 500 GIVING T1
JOIN part, T1 WHERE part# = part# GIVING T2
PROJECT T2 OVER color GIVING answer
```

The query is created and executed within a command file via the commands avialable at the RETRIEVE level.

5.2 G(et)

Get a command file from disk into the workfile. A workfile is simply a command file in memory. If the current workfile is not empty then the user must decide whether or not to throwaway the current workfile before getting another.

5.3 S(ave)

Save the workfile as a command file on disk. If a previous command file was obtained using G(et) the user is asked if he wishes to save the workfile with the same name. If a file already exists on the disk with the same name, the user must decide whether or not to destroy the file on disk before saving the workfile. If there is no room on the disk an error message is generated.

5.4 E(dit)

Typing "E" at the RETRIEVE level causes the following prompt-line to be displayed:

```
Edit ops: I(nsert), D(elete), B(egin), P(age), Q(uit) --->
```

Edit is used to create and modify command files while they are in the workfile. After execution of each Edit command except P(age) the first 20 or fewer lines of the command file is displayed on the screen, preceded by a line number.

A. I(nsert)

Insert one or more lines into the workfile. If a workfile exists then the user is prompted for a line number after which the new lines should be entered. An entry of 0 indicates before the first line and a line number greater than the last line number indicates after the last line. If a line in the file is specified, that line is displayed at the top of the screen and the user allowed to insert lines until only a return is entered. The workfile is renumbered after the insertion.

B. D(elete)

Delete a line from the workfile. The workfile is renumbered after the deletion.

C. B(egin)

Display the first 20 or fewer lines of the workfile.

D. P(age)

Display 20 or fewer lines starting at a particular line number of the workfile.

E. X(ecute)

Execute the command file in the current workfile. If syntax errors are found in the file, they are reported to the user. This causes execution to be aborted, however, the user can indicate that the syntax of the rest of the file is to be checked for syntax errors while ignoring the command in which the error occurred.

Each query of the command file is executed and the result is put into the temporary relation specified by the user in the query. If the query is determined to do nothing, such as the union of a relation with itself, then the query is not executed and this fact is reported to the user. Currently only the low-level procedure calls necessary to perform each query are output. In addition, only the structure charts (Appendix C: Figure C-21) for the security measures to be implemented have been designed.

F. D(isplay)

Display the contents of a relation on the screen. Again, only the structure charts (Appendix C: Figure C-22) for the security measures to be implemented have been designed.
When quiting the RETRIEVE level, the user must decide whether or not to throwaway the current workfile, if one exists.

UPDATE PROCEDURES * SECTION 6

To perform any update functions to the Access Control List, type "U" at the system level. The following prompt lines are displayed:

Enter the Relation Name ==>

```
Enter Owner-id              ==>
```

Here the user should enter any combination of 1-15 nonblank alphabetic, numeric characters or the following special characters: ".", "-", "_", "/". The relation name entered should be defined in the DDL, otherwise the following prompt is displayed:

```
Relation Name => XXXXXXXXXX not defined in DDL.
Please consult with the DBA!!!!!!!
```

In addition the following error message is written into the ERROR.DATA file:

```
ERROR => 5
ID    => USER-ID
RELATION NAME => Relation Name
```

and processing returns to the System Option level. In a similar fashion the user can enter any combination of 10 nonblank alphabetic (A-Z), numeric (0-9) characters. This combination of characters uniquely identifies the owner of the relation. If permission is granted the following prompts are displayed:

```
Enter USERNAME ==>
```

Here the user should enter the name of the user to be granted access to the relation. This should be entered in the following manner: first name, blank, middle initial, blank, last name;

```
Enter Access Rights: (R)ead, (M)odify, (I)nsert,
```

```
(D)elete, (E)xecute.
Please Enter (example: RMD) ==>
```

Here the user should enter a "R", "M", "I", "D", "E", or any combination, in any order as long as there are no duplicates entered. In addition, no more than five access rights may be assigned at a time and no access right can be separated by a comma or blank. Once this has been done the following prompt is displayed:

```
Please enter USER-ID ==>
```

Here the user will enter the user's identification name. However, this code will need to be removed once the user-id can be retrieved from a relation. This will be necessary to ensure that the user identification names are protected.

If a valid user-id is obtained then the following prompt is displayed:

```
Please select ONE of the following.
(A)dd      ==> Adds user-id and access rights.
(D)elete   ==> Deletes user-id and access rights.
(R)eplace  ==> Replaces user-id and access rights.
(Q)uit      ==> Terminates

Enter A, D, R, or Q ==>
```

Here the user will select one and only one of "A", "D", "R", or "Q". If the user selects a "A", "D", or "R" then the user-id extracted from the relation and the access rights to be granted are either added, deleted, or replaced in the Access Control List. If anything other than an "A", "D", "R", or "Q" is entered the following prompt lines are displayed:

```
Invalid option: PLEASE TRY AGAIN!!!
Enter A, D, R, or Q ==>
```

If "Q" is entered then processing returns to the calling module. On the other hand, if an invalid user-id is obtained from the relation tuple then the following error message is written into the ERROR.DATA file:

```
ERROR => 7
ID    => USER-ID
RELATION NAME => Relation Name
INVALID USERS NAME => Username
```

and processing returns to the calling module.

Once a user-id and access rights have been added, deleted or replaced the following prompt is displayed:

```
Do you wish to Enter more USERS & ACCESS RIGHTS
to this relation? (Y/N) =>
```

If "Y" is entered then the user is again prompted with all the prompts previously mentioned under the permission granted area. On the other hand if "N" is entered or permission is denied processing continues and the permission denied flag is checked. If this flag is set to true (meaning this user did not enter the correct relation name/owner-id combination) then the following error message is written into the ERROR.DATA file:

```
ERROR => 6
ID     => USER-ID
RELATION NAME => Relation Name
```

Regardless of whether the permission denied flag has been

set, processing returns to the System Option level.
Note, all answers to yes or no questions must be answered with either a "Y" or "N".

COMMAND SUMMARY * SECTION 7

7.1 E(dit)

"E" places the user in a level of the system called Edit. This section of the system contains commands used primarily for maintenance of relations. The options allowed are listed as follows:

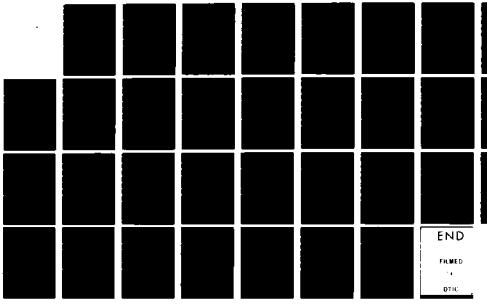
1. Insert -- allows the user to add tuples to a relation
2. Delete -- allows the user to remove tuples from a relation
3. Modify -- allows the user to make changes to tuples of a relation
4. Copy -- allows the user to insert tuples from relation into another
5. Transfer -- allows the user to move tuples from one relation into another relation

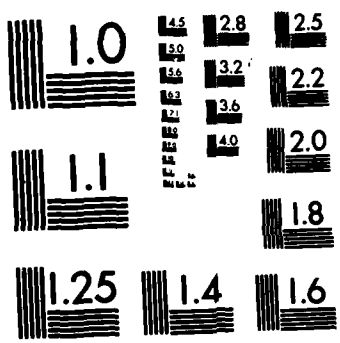
See section 4 for more detail.

7.2 R(etrieve)

This command allows the user to formulate and execute relational queries on the database relations, and display the results. The user is allowed to formulate queries that perform the select, project, or join operation. In addition, the user can perform the union, intersect, divide, and difference operations. A workfile is used to hold queries, and commands are included perform various functions on this workfile.

AD-A124 840 THE ANALYSIS AND DESIGN OF A COMPUTER SECURITY/RECOVERY 3/3
SYSTEM FOR A RELA..(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W P REED
UNCLASSIFIED 17 DEC 82 AFIT/GCS/EE/82D-26 . F/G 9/2 NL





7.3 I(nventory)

Typing "I" at the system level will cause a list of the relations which have been defined and attached to be displayed on the CRT. Note, all information but the classifcaton levels of the attributes are listed.

7.4 A(ttach)

This command allows the user to attach a relation. Currently, only an attach flag is set in the relation definition. However, even though a relation has been attached, unless the user's identification name is found in the Access Control List for this relation any type of access will be denied.

7.5 Q(uit)

This command allows the user to exit from the database system. During this process, all relation and domain definitions are written out to disk, and a message is displayed when this process has completed. In addition, the Access Control List which contains relation names, owner-ids, and lists of user-ids with associated access rights are written out to disk.

7.6 U(pdate)

This command allows the owner of a relation to add, delete, or replace a user-id and the associated access rights granted to this user provided the correct relation name/owner-id combination has been entered. In addition, any relation

name/owner-id combinations that do not exist in the Access Control List are added and any illegal transactions are written into an error file which is monitored by the DBA.

CHANGING THE SYSTEM * SECTION 8

Please refer to sections 3.3.1 and 3.3.2 of the UCSD Pascal Version II.0 reference manual and current program listings before trying to change the system.

The program is currently divided into a main body segment, four segment procedures and a unit. Each segment is contained in a dummy program in order to permit seperate compilation. The unit contains all types, variables, and procedures which are global to more than one segment. Thus by including the unit in each dummy program, access is allowed to those elements. The format for each segment procedure is:

```
PROGRAM dummy-name;
USES COMMON;  (*the unit is named COMMON*)
SEGMENT PROCEDURE name(parameter-list);
  Local types, variables, and procedures;
BEGIN
  body of name;
END; (*name*)
BEGIN
END.  (*dummy name*)
```

Since the segments are seperately compiled, the parameter list of the segment procedure, must contain all global variables accessed or modified by the procedure. The program or segment procedure whcih calls each segment procedure must have a dummy segment procedure with the same name, so that it may compile properly. The format for the main body segment is:

```
PROGRAM main;
USES COMMON;
  local labels,types,constants, and variables;
SEGMENT PROCEDURE namel( --- );
BEGIN
END; (*namel*)
SEGMENT PROCEDURE name2( --- );
BEGIN
END; (*name2*)
.
.
.
other local procedures;
BEGIN
  body of main;
END. (*main*)
```

The format for segment procedures which use other segments is the same as the previous format for a segment procedure except dummy segment procedures are included as local procedures.

Each segment procedure has a particular segment number from 11 to 15 associated with it. The main body segment has number 1 and the unit has number 10. Other numbers are for Pascal use only. The way numbers are assigned to the segment procedures is in first compiled, first numbered order. Thus, in the above format for the main body segment namel would be assigned 11, name2 assigned 12, etc. Therefore, in each dummy program used to define a segment procedure an appropriate number of dummy segments must exist before the defined segment to ensure that the segment count is the same. Thus, for example, the format for segment name2 would be:

```
PROGRAM dummy name;
USES COMMON;
SEGMENT PROCEDURE dummy namel;
BEGIN
END; (*dummy namel*)
```

```
SEGMENT PROCEDURE name2( --- );
  local labels, types, etc.
BEGIN
  body of name2
END; (*name2*)
BEGIN
END.  (*dummy name*)
```

The unit -- COMMON -- is compiled and placed in the system library using the librarian program, LIBRARY.CODE. (See section 4.2 of the UCSD Pascal manual.) When each program is compiled, the unit is retrieved from the system library and used in the program, however, each program must still be linked with the system library in order to bind the external variable and procedure references into the unit. After each program is compiled and linked, then the librarian may be used to put all the segments together into one code file. Each code file containing the segment is retrieved and linked into the proper space using its assigned segment number into the overall file.

If a particular segment, including the main body segment, is to be changed then the steps to be followed are:

1) Change the source code for the segment.

2) Compile the program containing the segment.

3) Link the code file to the system library.

4) Using the librarian create a new overall file passing all unchanged segments to the new file and linking in the changed segment.

If the unit has to be changed then after compiling it and placing it into the system library, each program must be recompiled, linked to the system library, and then put together with the librarian.

APPENDIX C

CS/RS Structure Chart Documentation

This appendix consists of two sections documenting the modular design of the CS/RS software. The first section is a list identifying all the modules in the software design. The '*' that appears to the right of the module's title identifies that module as a common module called by more than one calling module. The second section is a set of structure charts showing the interrelated structure of the modules within the subsystem. The module numbers in each section are used as a cross-reference between both documentation sections. The program listings for this subsystem are contained in Volume II of this thesis. The inputs and outputs to each module can be found in the module headers of the source code.

Figures C-1 to C-7 pertain to the DDL Facility program.

Figures C-8 to C-22 pertain to the DB Main program.

DDL Facility Program

1.0	DBMGR	
1.1	INVALID_ERROR	*
1.2	SETUP	*
1.3	ACL_SETUP	*
1.3.1	GET_TREE	
1.3.1.1	ADD_REL_OWNERID	
1.3.1.1.1	RELATTACH	
1.3.1.2	DO_OPTION	*

1.3.1.2.1	INSERTNODE	
1.4	DBMGR OPTIONS	
1.4.1	DEFINE	
1.4.2	INITIALIZE	
1.4.3	REMRELATION	
1.4.3.1	SEARCH_DDL	*
1.4.3.2	ACT_DELETE	
1.4.3.2.1	DEL	
1.4.4	QUIT	*
1.4.4.1	PUTREL	
1.4.4.2	PREORDER	
1.4.5	INVENTORY	

DB PROGRAM

1.0	DB	
1.1	DB_ACCESS_MONITOR	
1.1.1	GETID	
1.1.2	CKIDACCESS	
1.1.2.1	EXCOMSTRING	
1.1.2.2	EXTRACT ID STRING	
1.4	DB OPTIONS	
1.4.1	UPDATE	
1.4.1.1	GET_RELNAME_OWNERID	
1.4.1.3	AC_TREE_SEARCH	
1.4.1.5	ADD_REL_OWNERID	
1.4.1.5.1	RELATTACH	
1.4.1.6	RETR_USERID	

1.4.1.6.1	GET_USER NAME	
1.4.1.6.2	EXCOMSTRING	
1.4.1.6.3	ID_EXTRACT	
1.4.1.6.4	SELECT_OPTION	
1.4.2	USEDIT	
1.4.2.1	VALIDINSERTCMD	
	VALIDID	*
	ACL_SEARCH	*
	CK_RIGHTS	*
1.4.2.2	VALIDDELETECMD	
1.4.2.3	VALIDMODCMD	
1.4.3	ATTACH	
1.4.6	RETRIEVE	

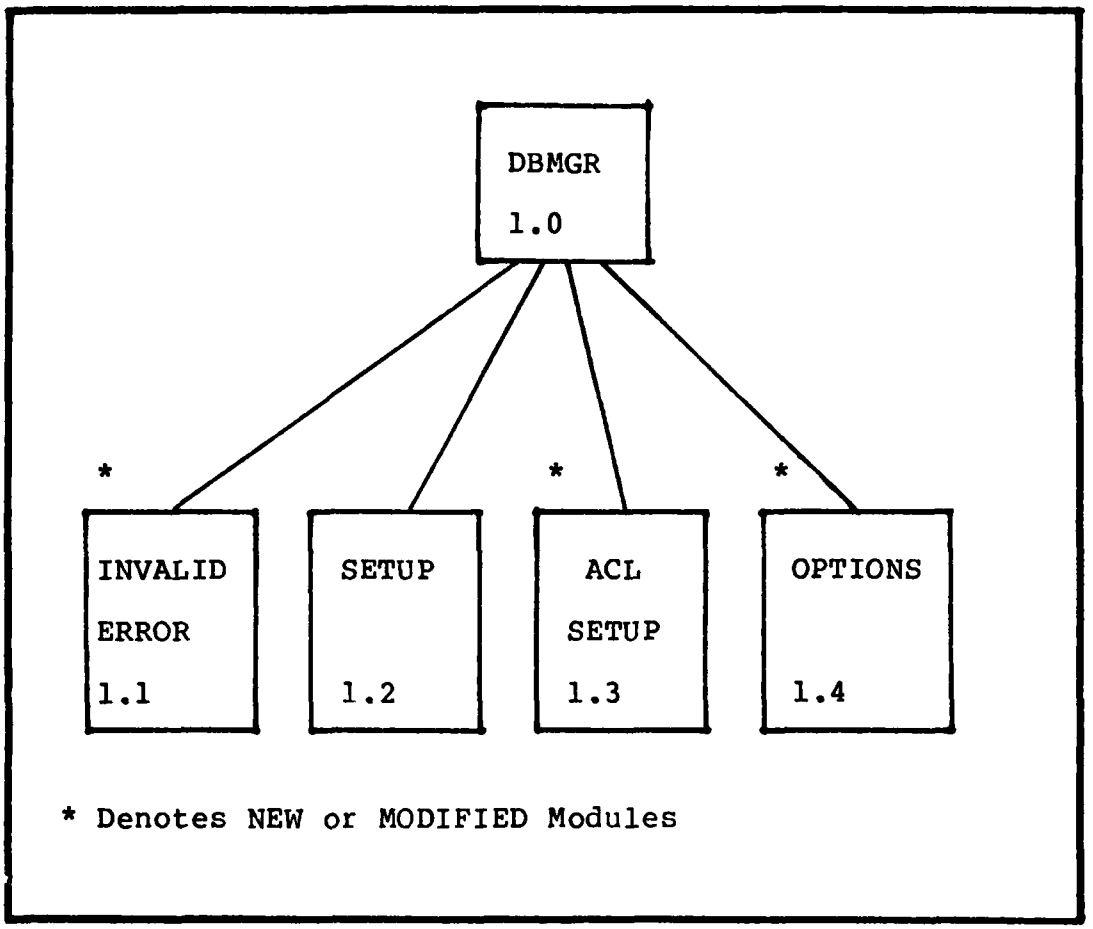


FIGURE C-1: DBMGR Module Level 1.0

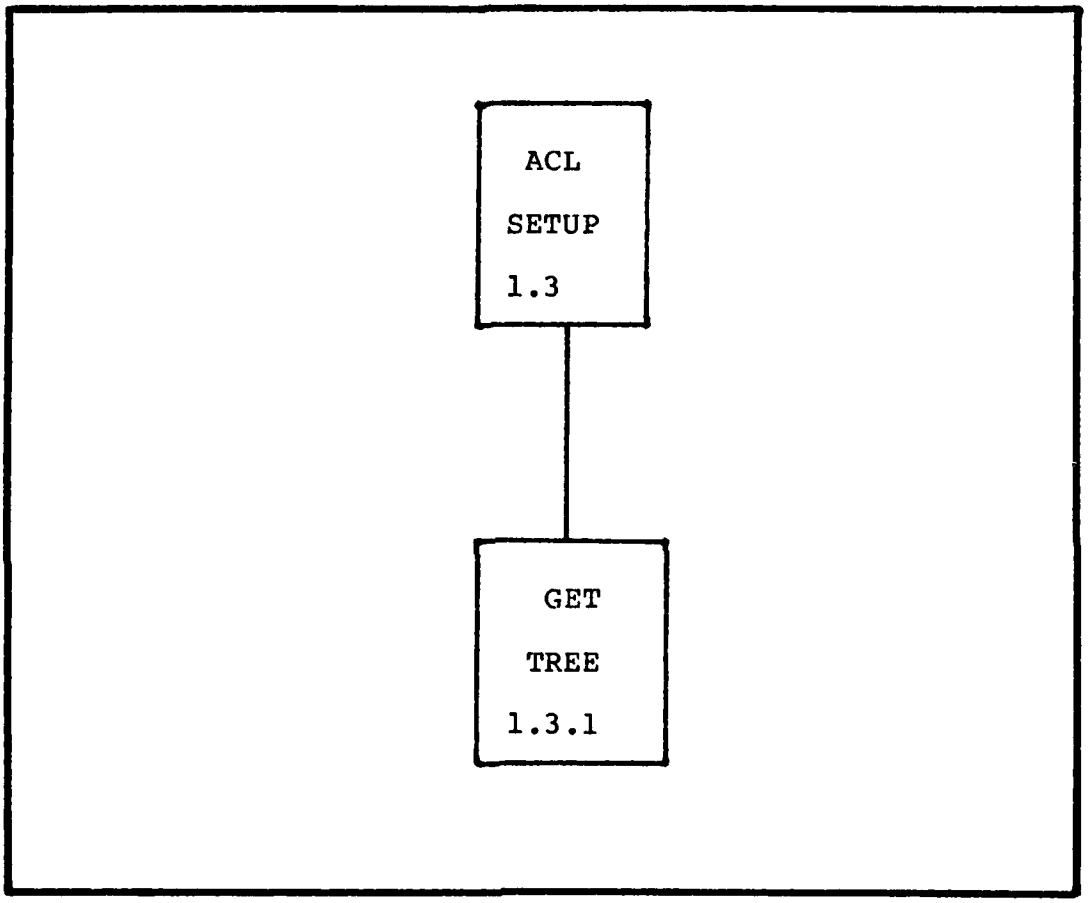


FIGURE C-2: ACL SETUP Module Level 1.3

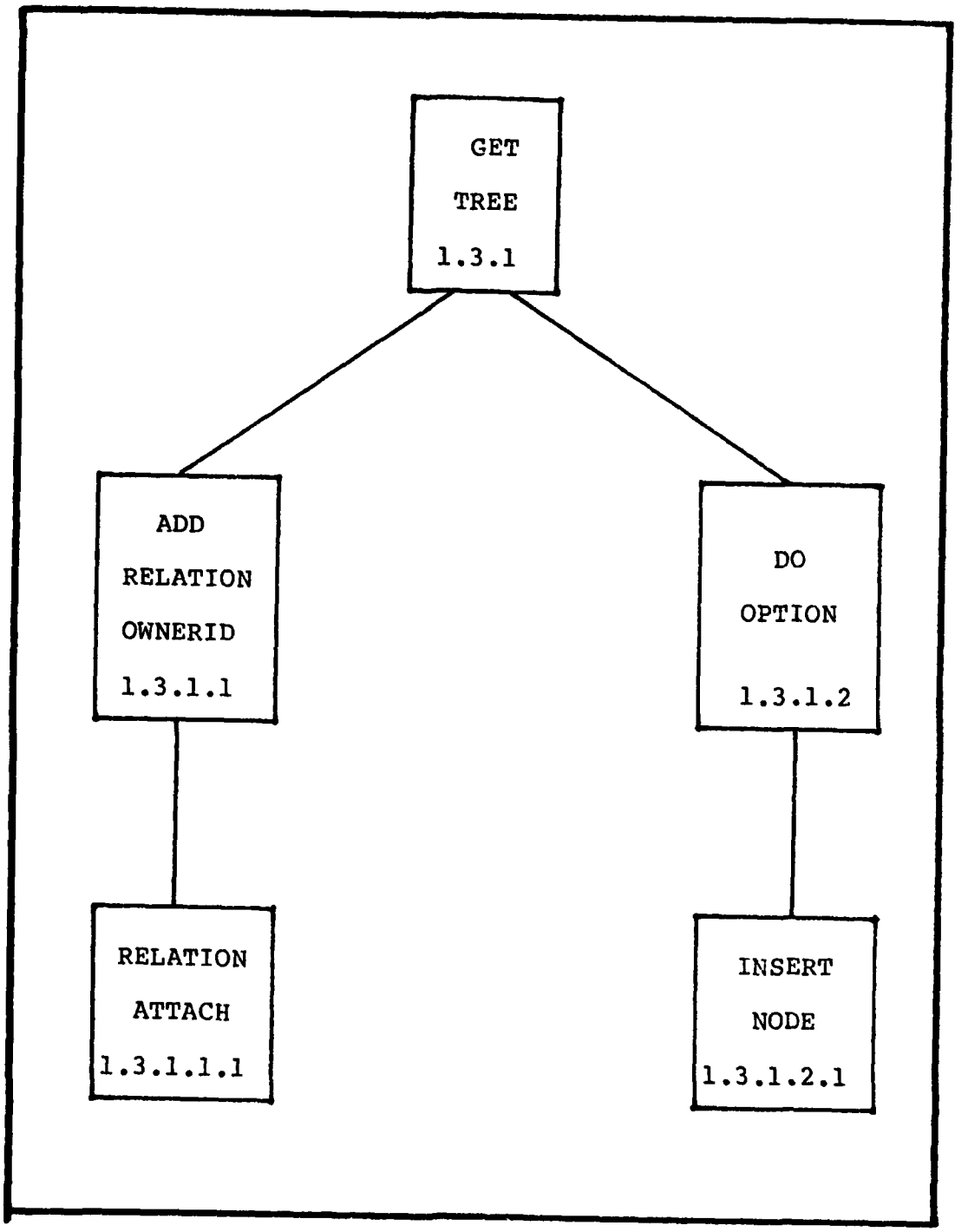


FIGURE C-3: GET TREE Module Level 1.3.1

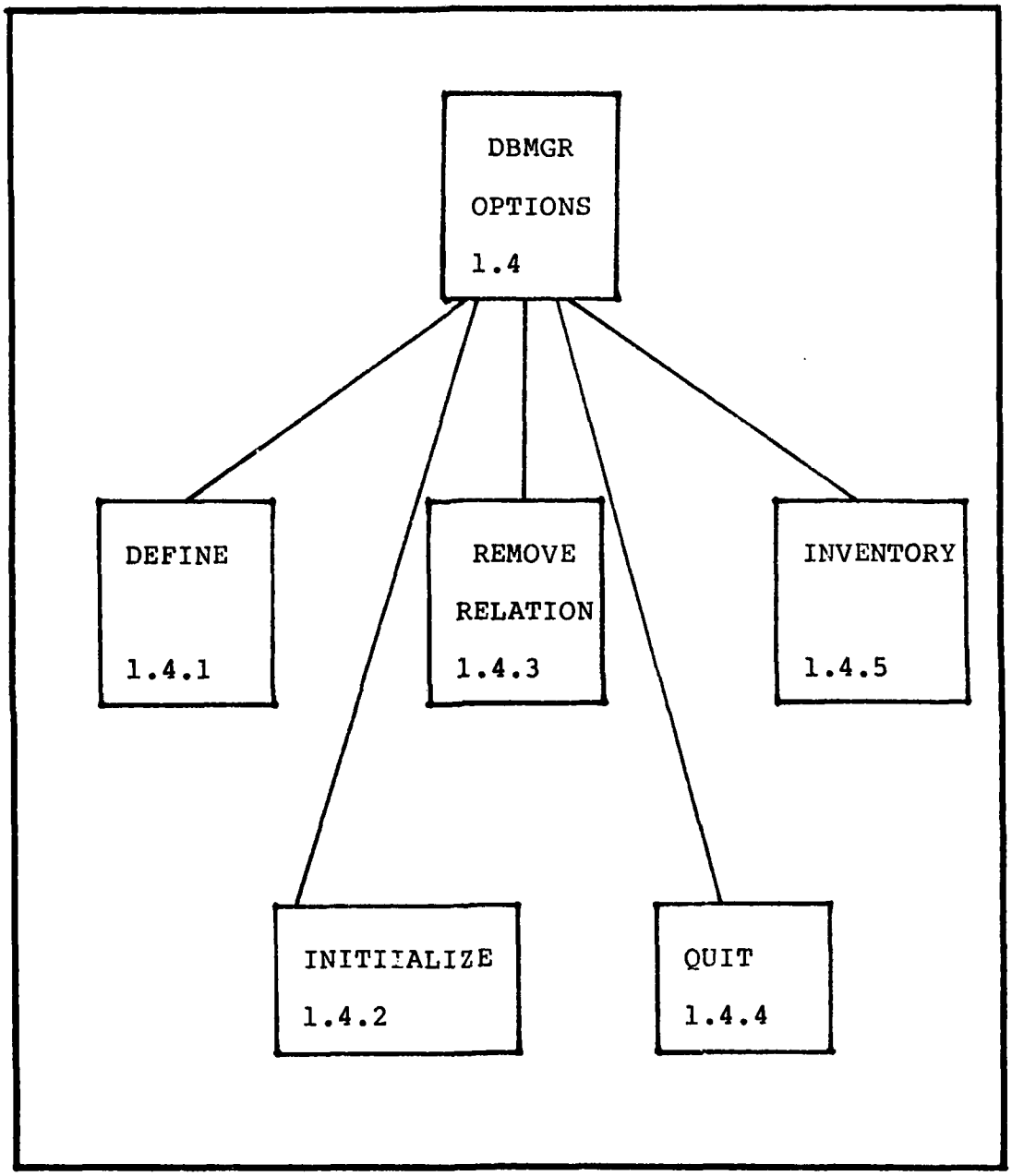


FIGURE C-4: DBMGR Options Level 1.4

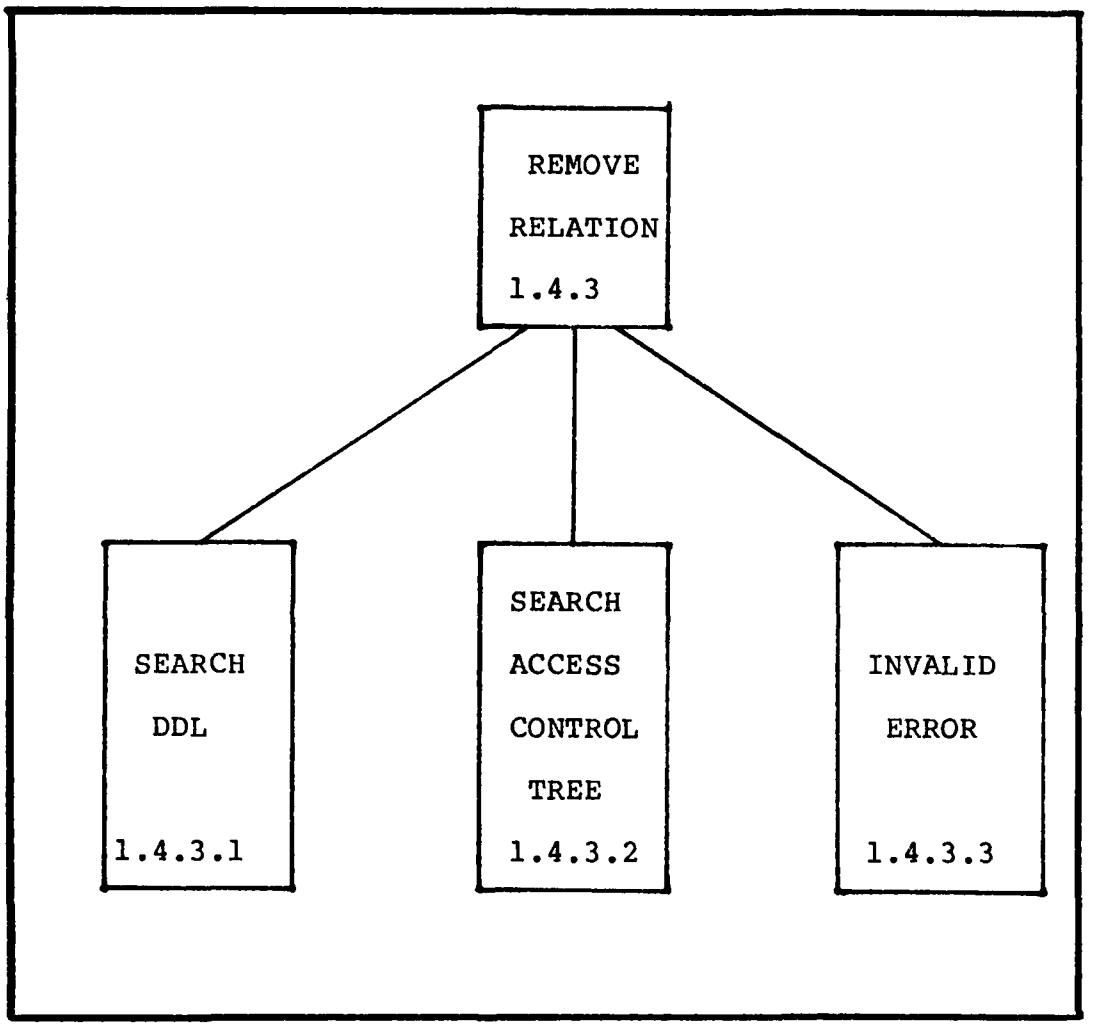


FIGURE C-5: REMOVE RELATION Module Level 1.4.3

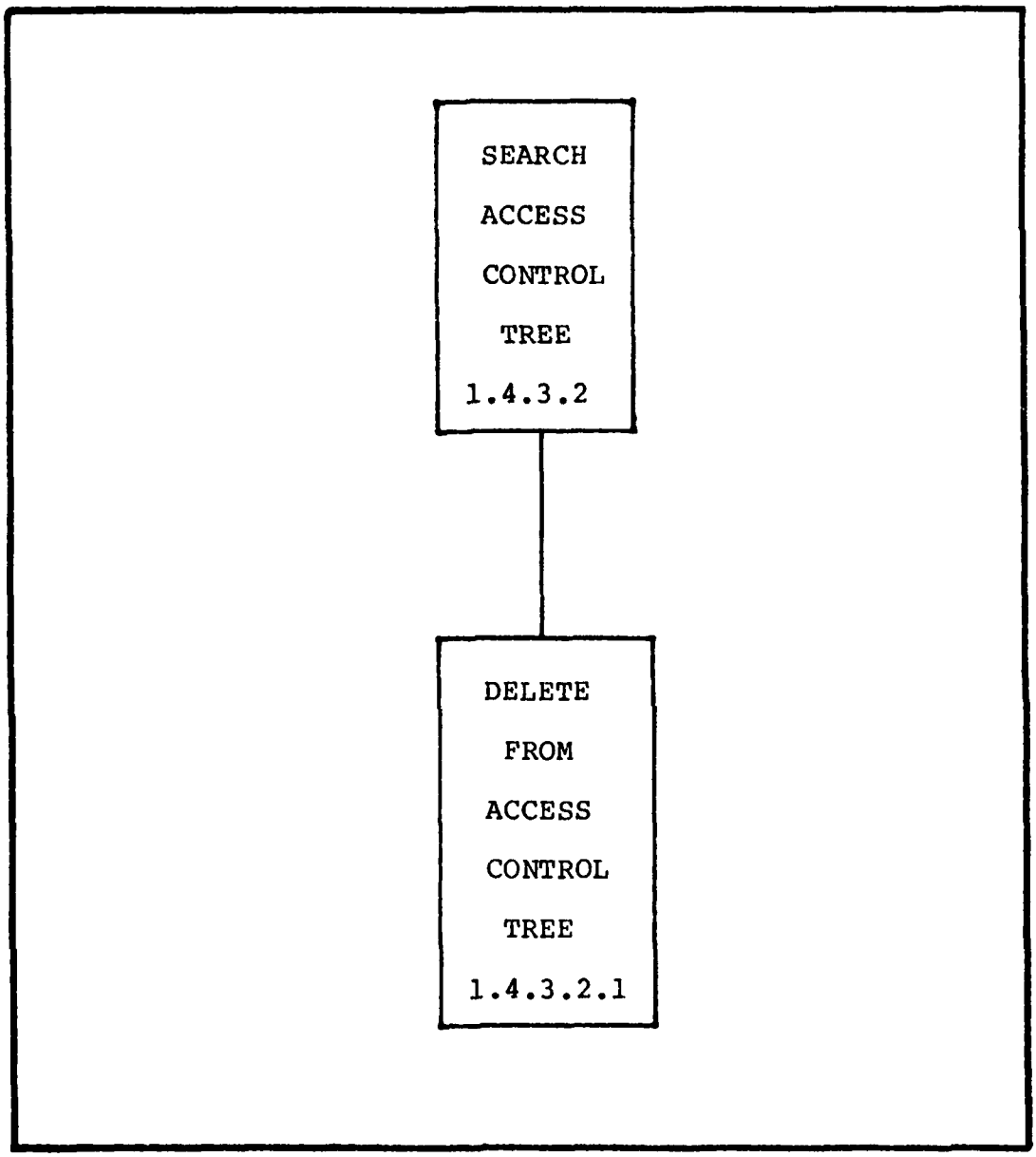


FIGURE C-6: ACT DELETE Module Level 1.4.3.2

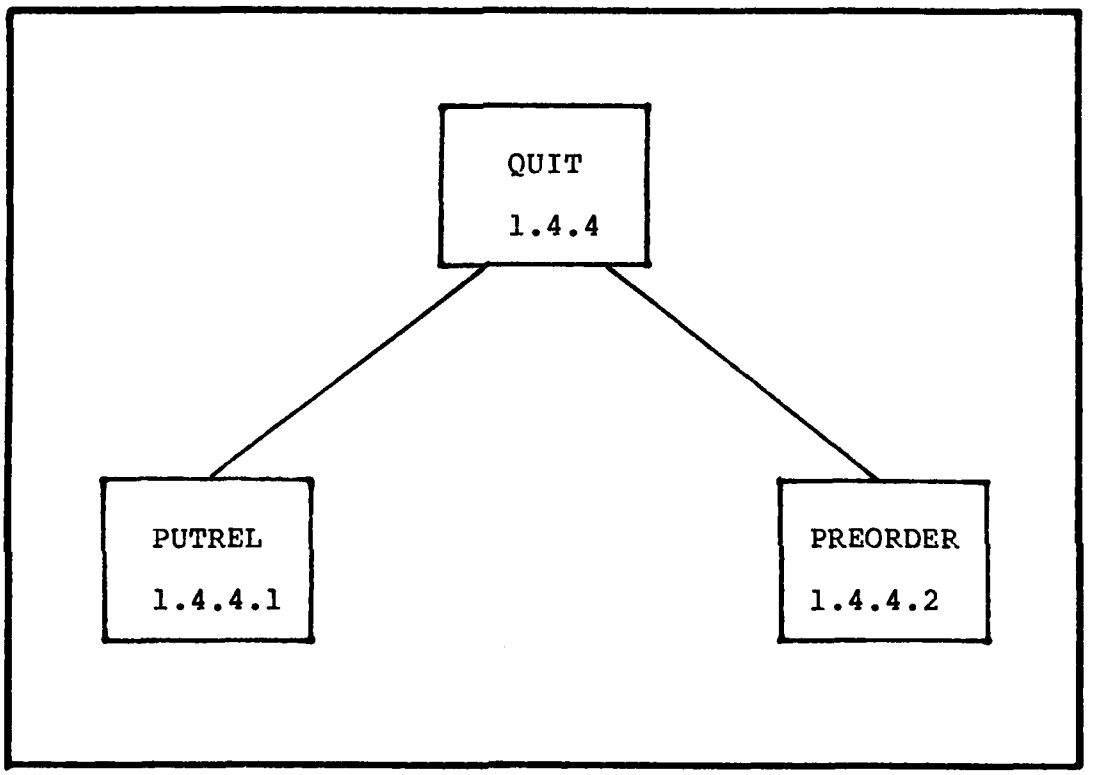


FIGURE C-7: QUIT Module Level 1.4.4

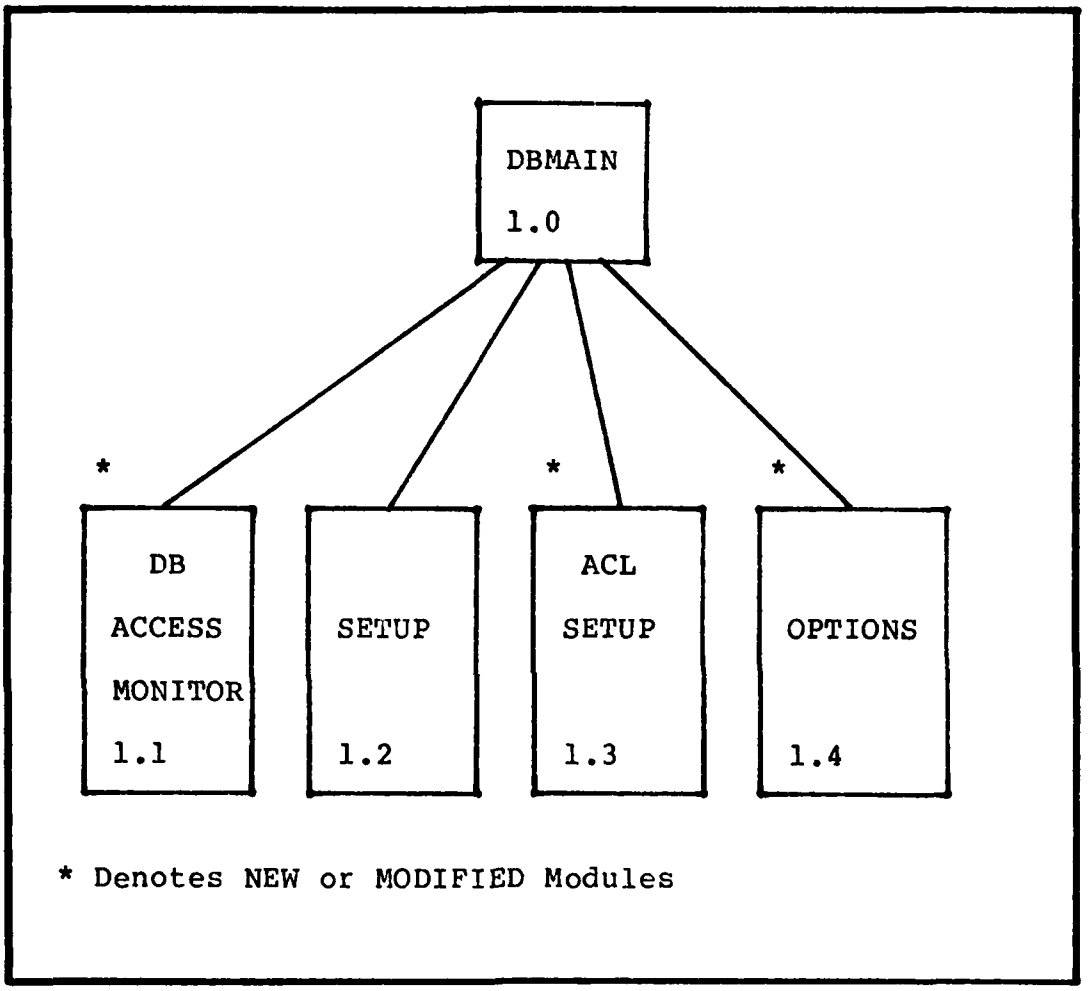


FIGURE C-8: DBMAIN Module Level 1.0

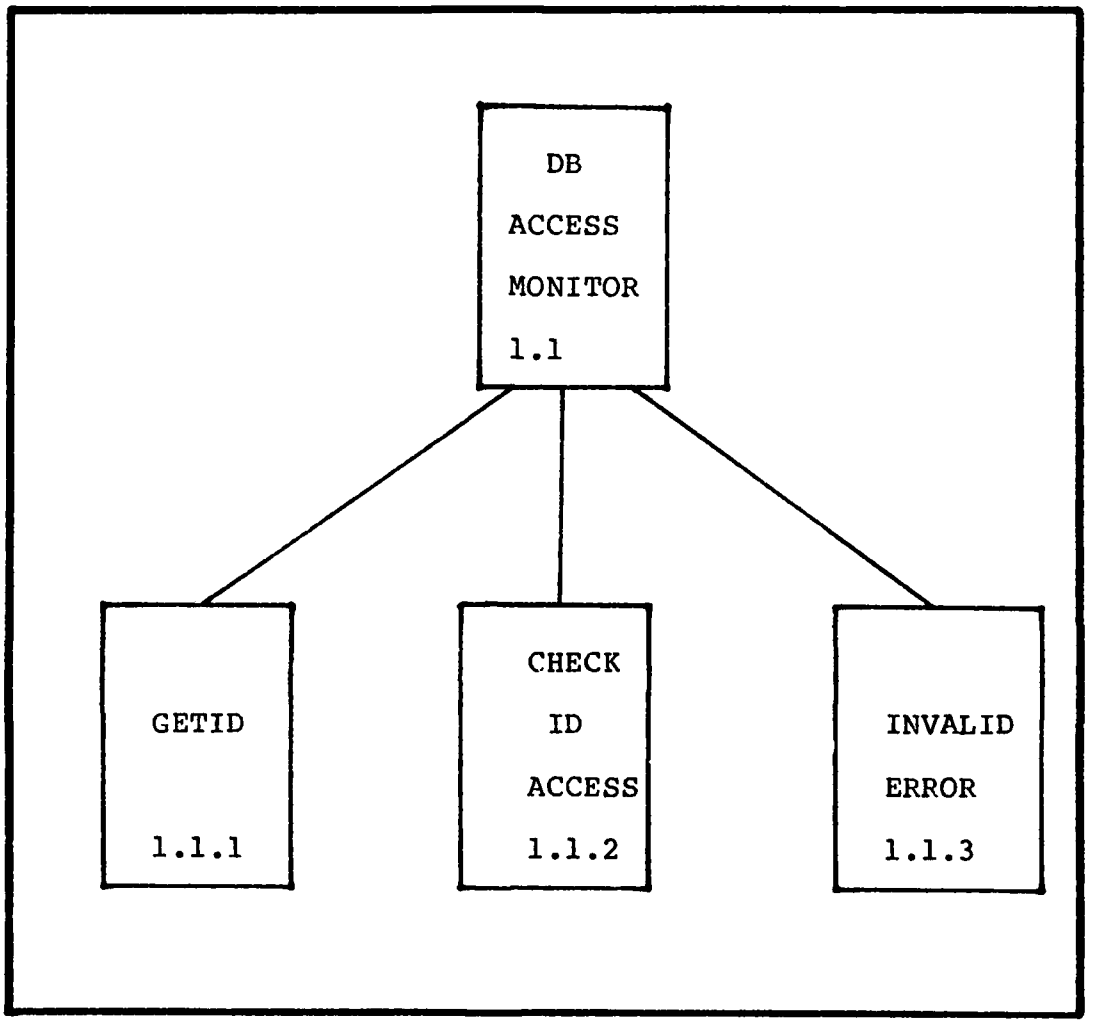


FIGURE C-9: DB ACCESS MONITOR Module Level 1.1

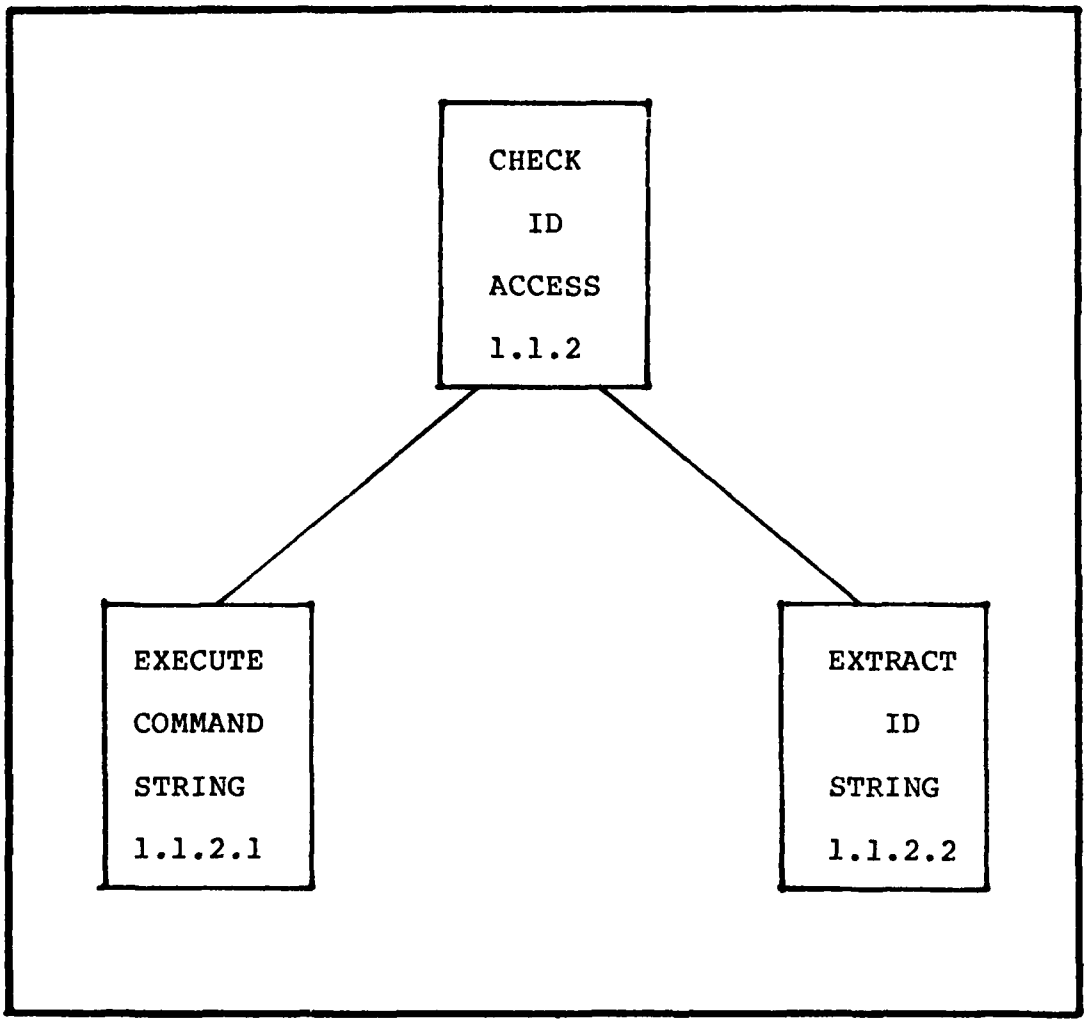


FIGURE C-10: CHECK ID ACCESS Module Level 1.1.2

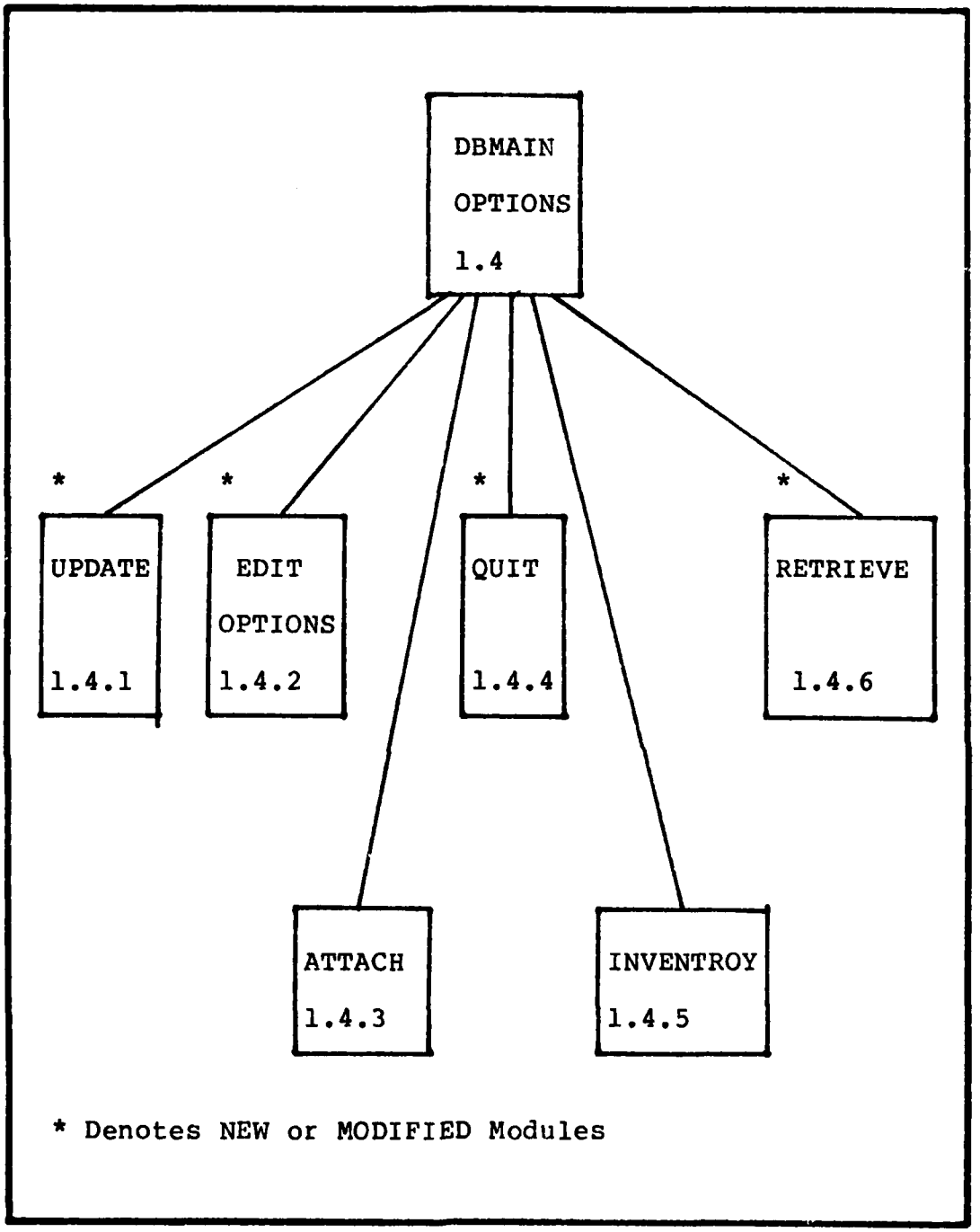


FIGURE C-11: DBMAIN OPTIONS Module Level 1.4

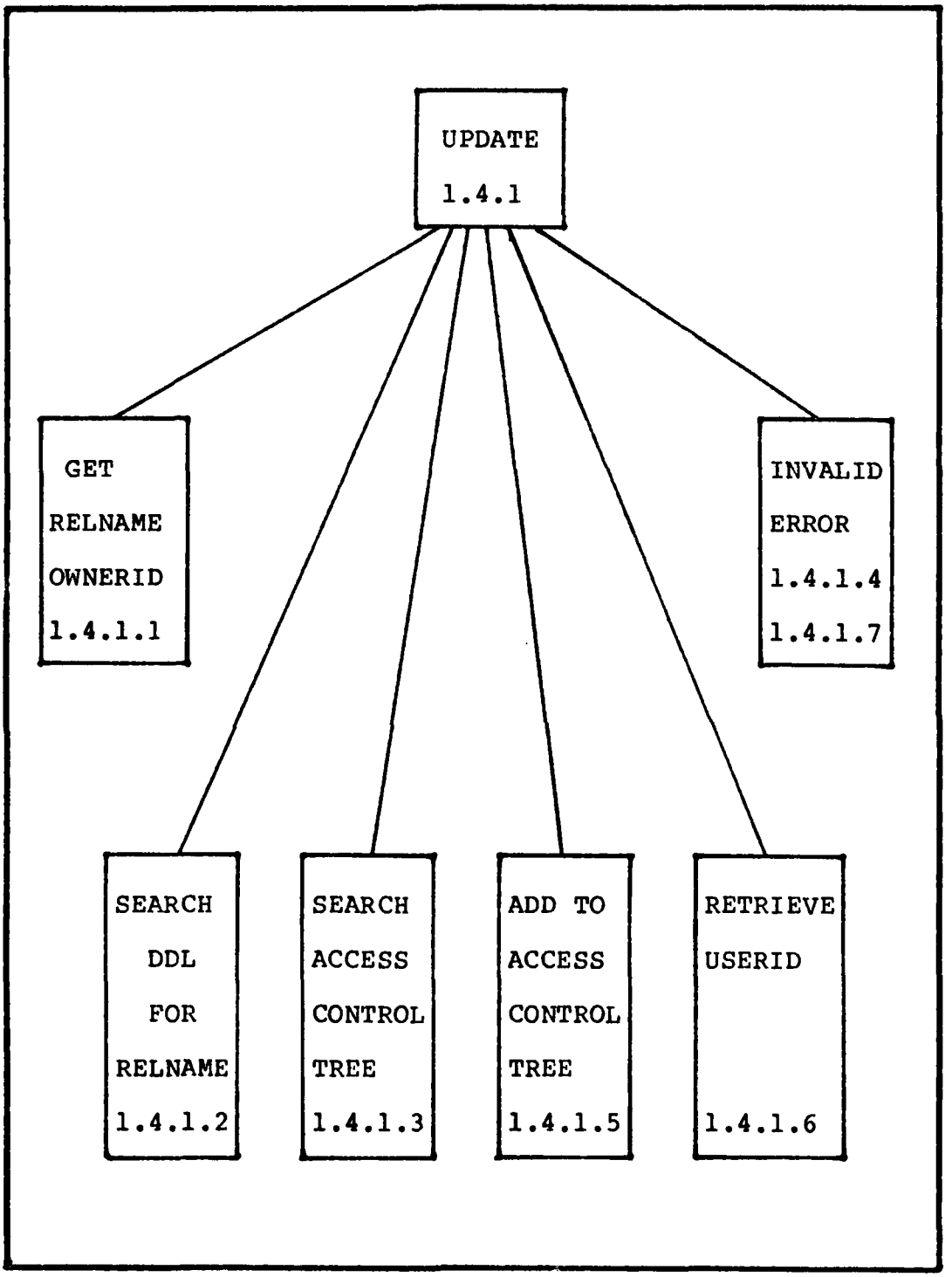


FIGURE C-12: UPDATE Module Level 1.4.1

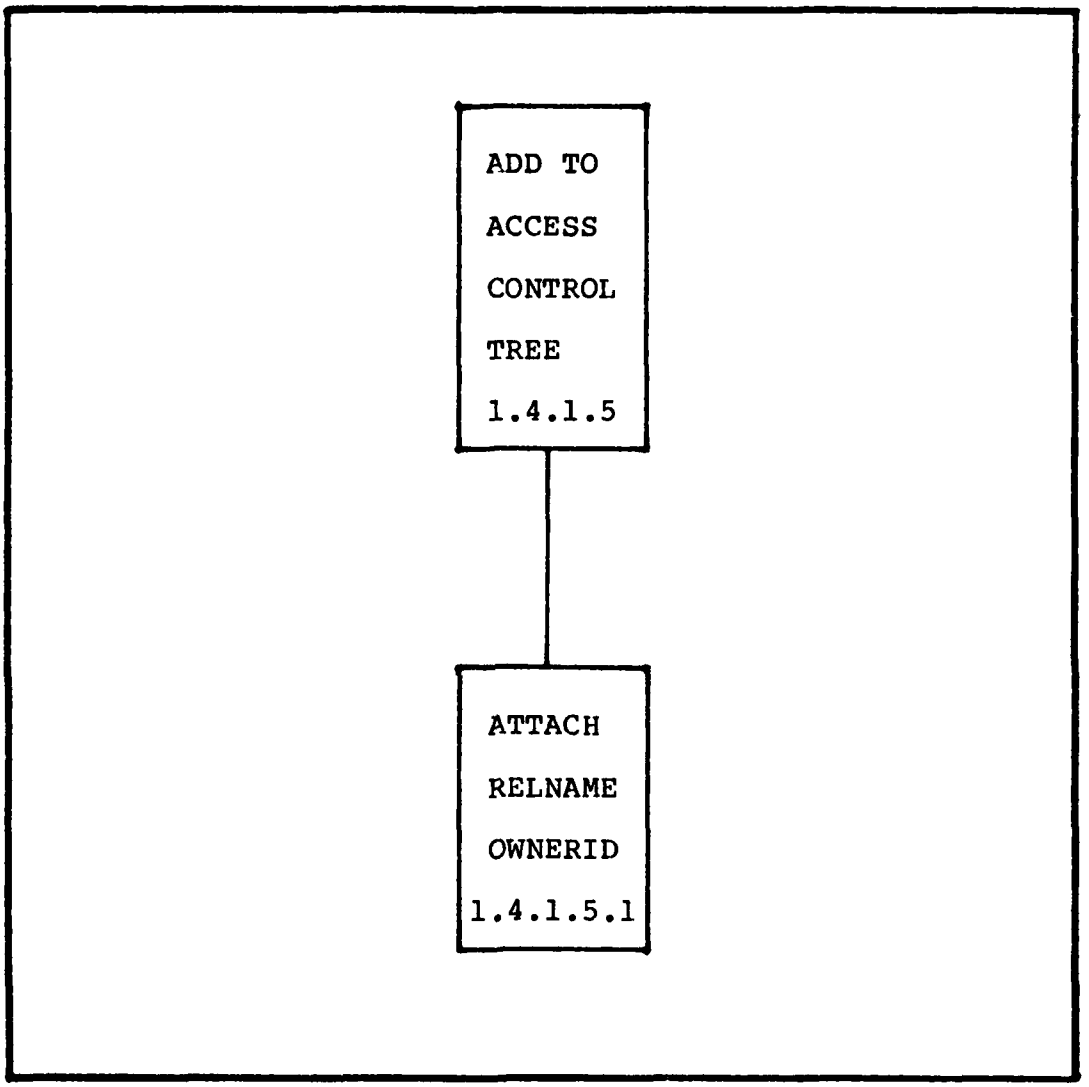


FIGURE C-13: ADD TO ACT Module Level 1.4.1.5

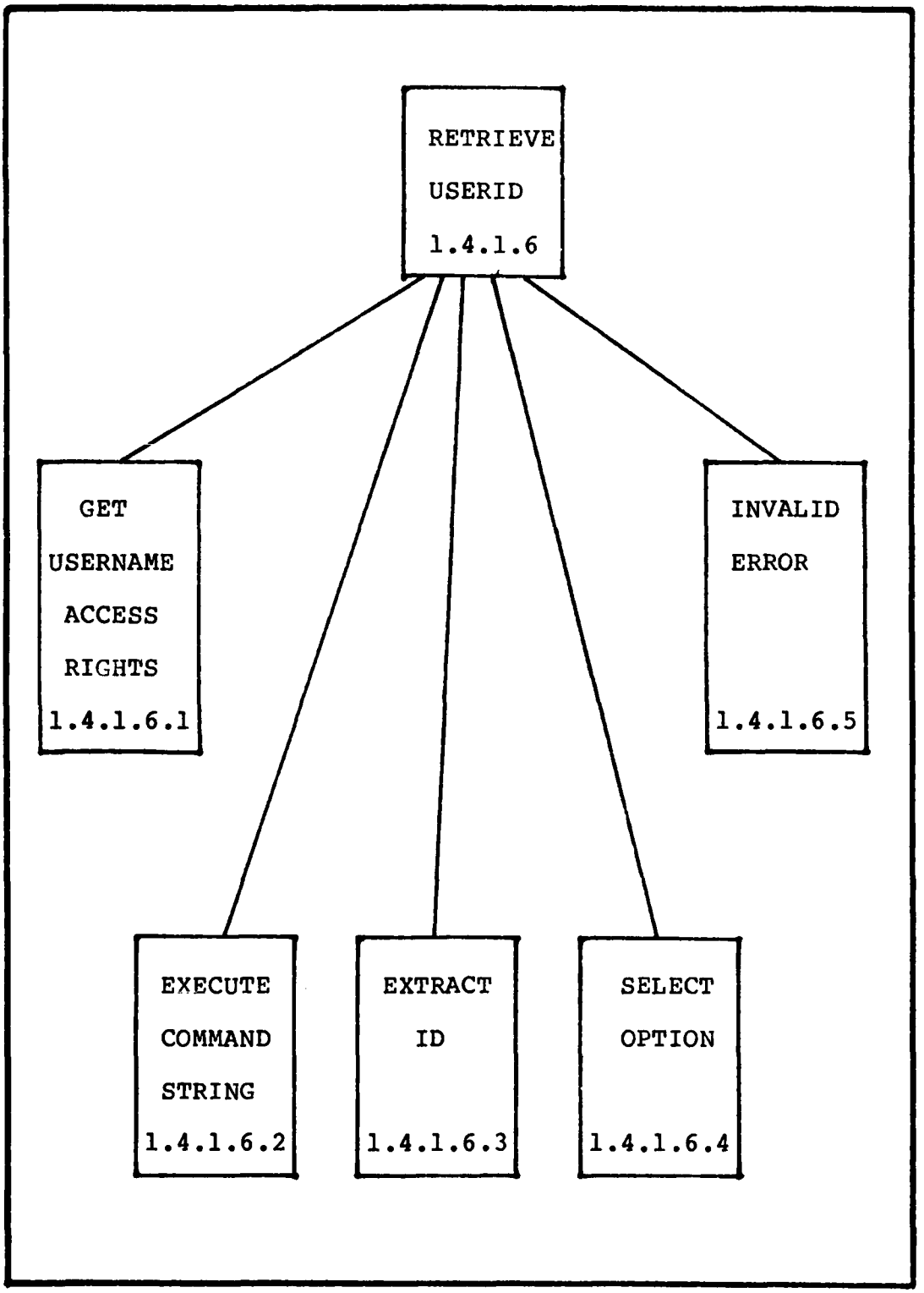


FIGURE C-14: RETRIEVE USERID Module Level 1.4.1.6

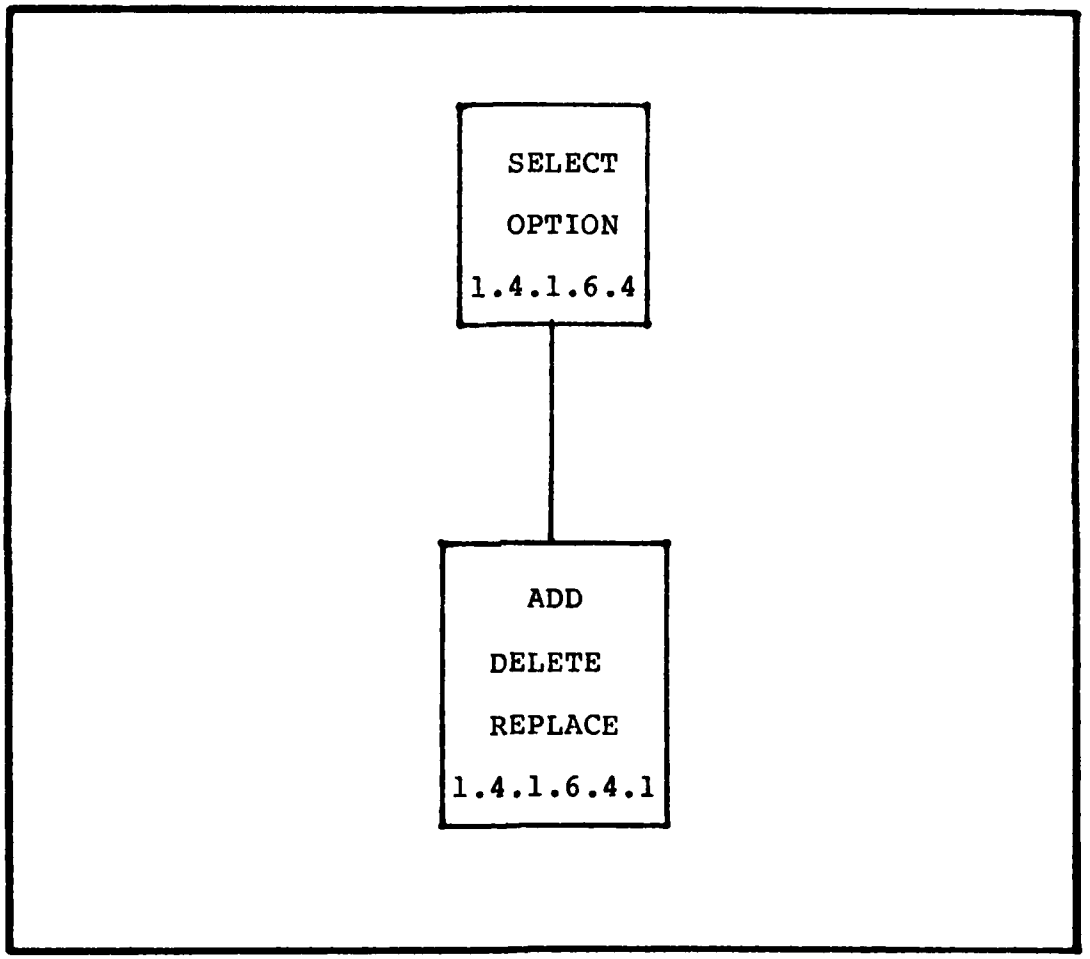


FIGURE C-15: SELECT OPTION Module Level 1.4.1.6.4

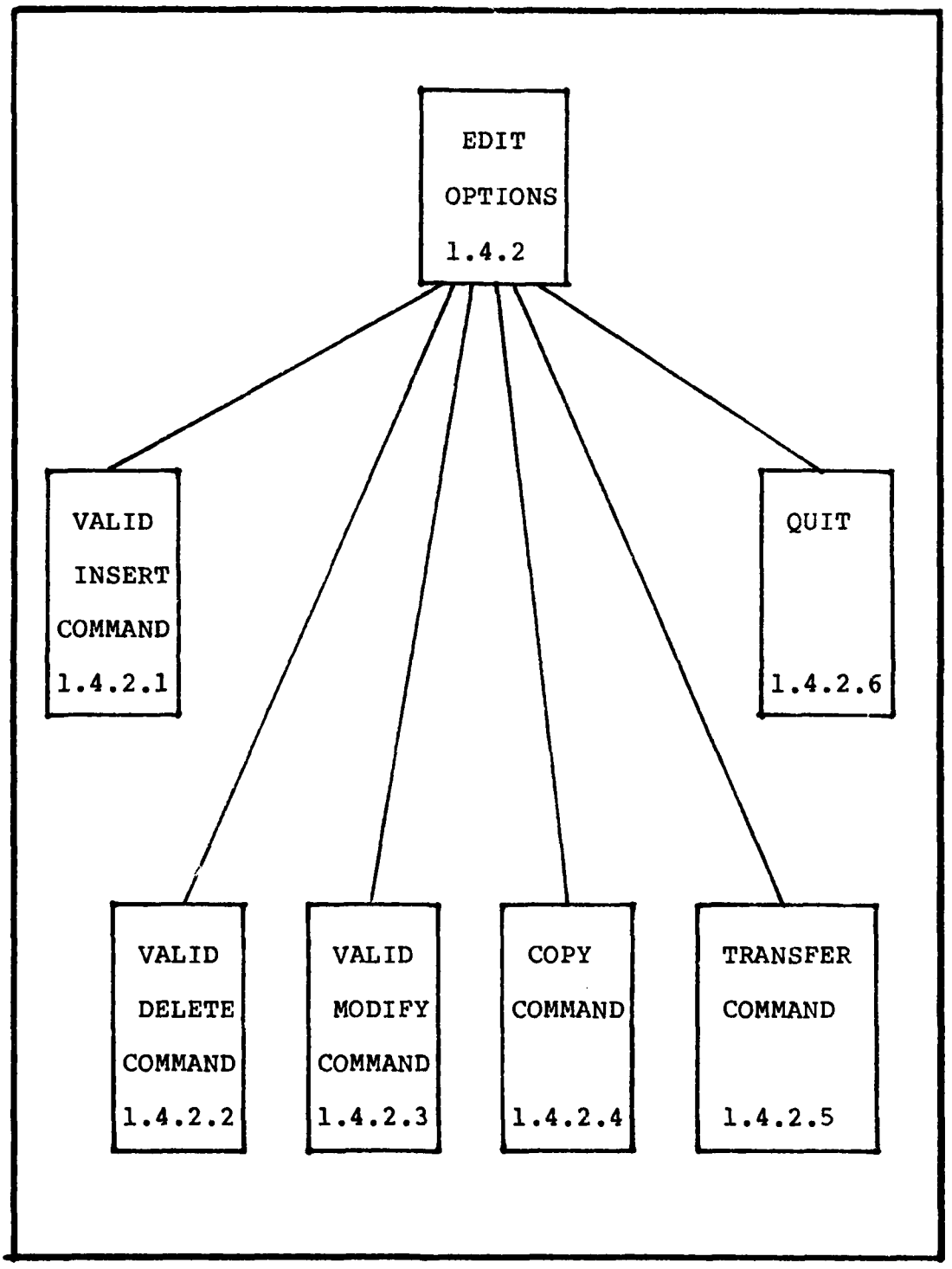


FIGURE C-16: EDIT Options Module Level 1.4.2

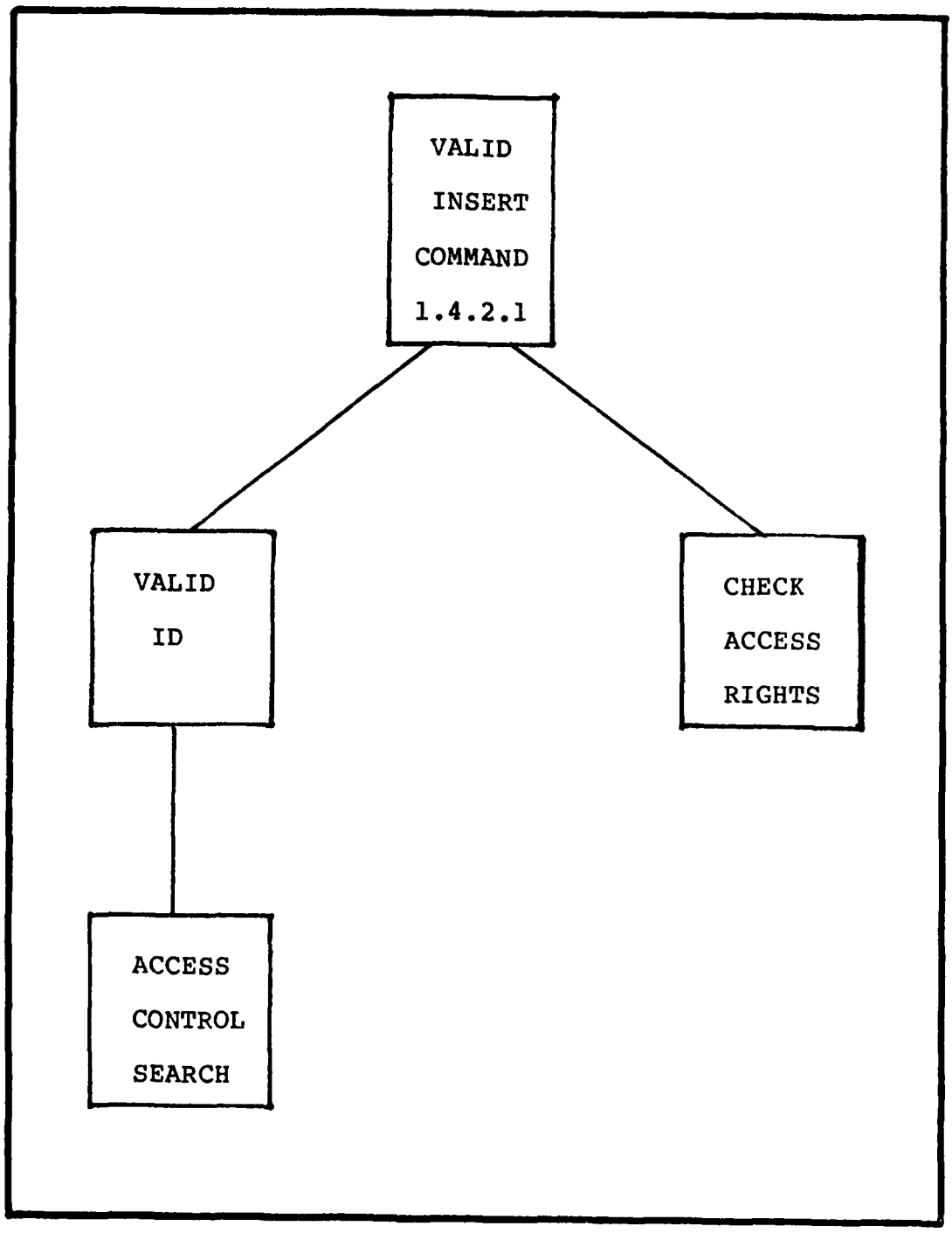


FIGURE C-17: VALID INSERT COMMAND Module Level 1.4.2.1

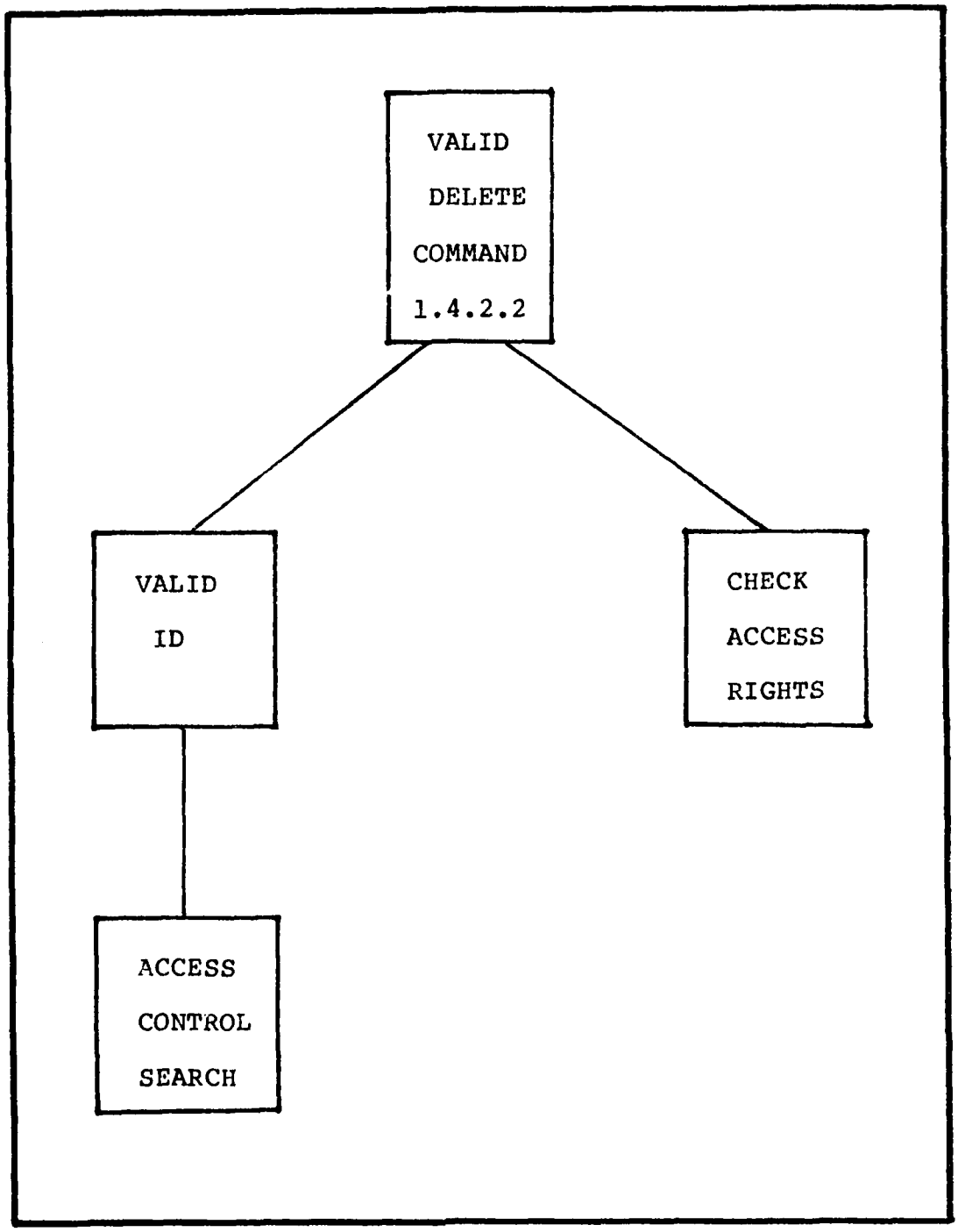


FIGURE C-18: VALID DELETE COMMAND Module Level 1.4.2.2

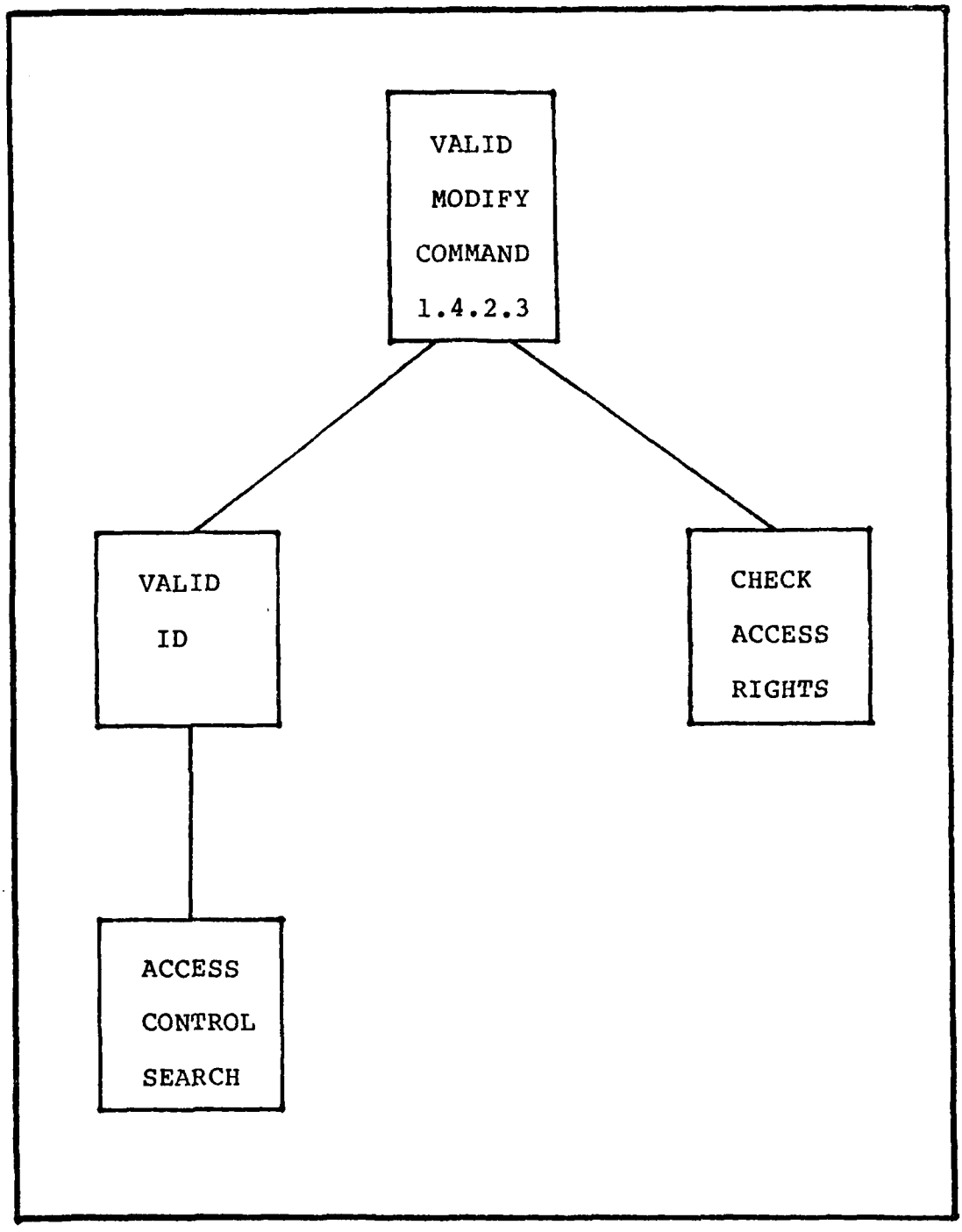


FIGURE C-19: VALID MODIFY COMMAND Module Level 1.4.2.3

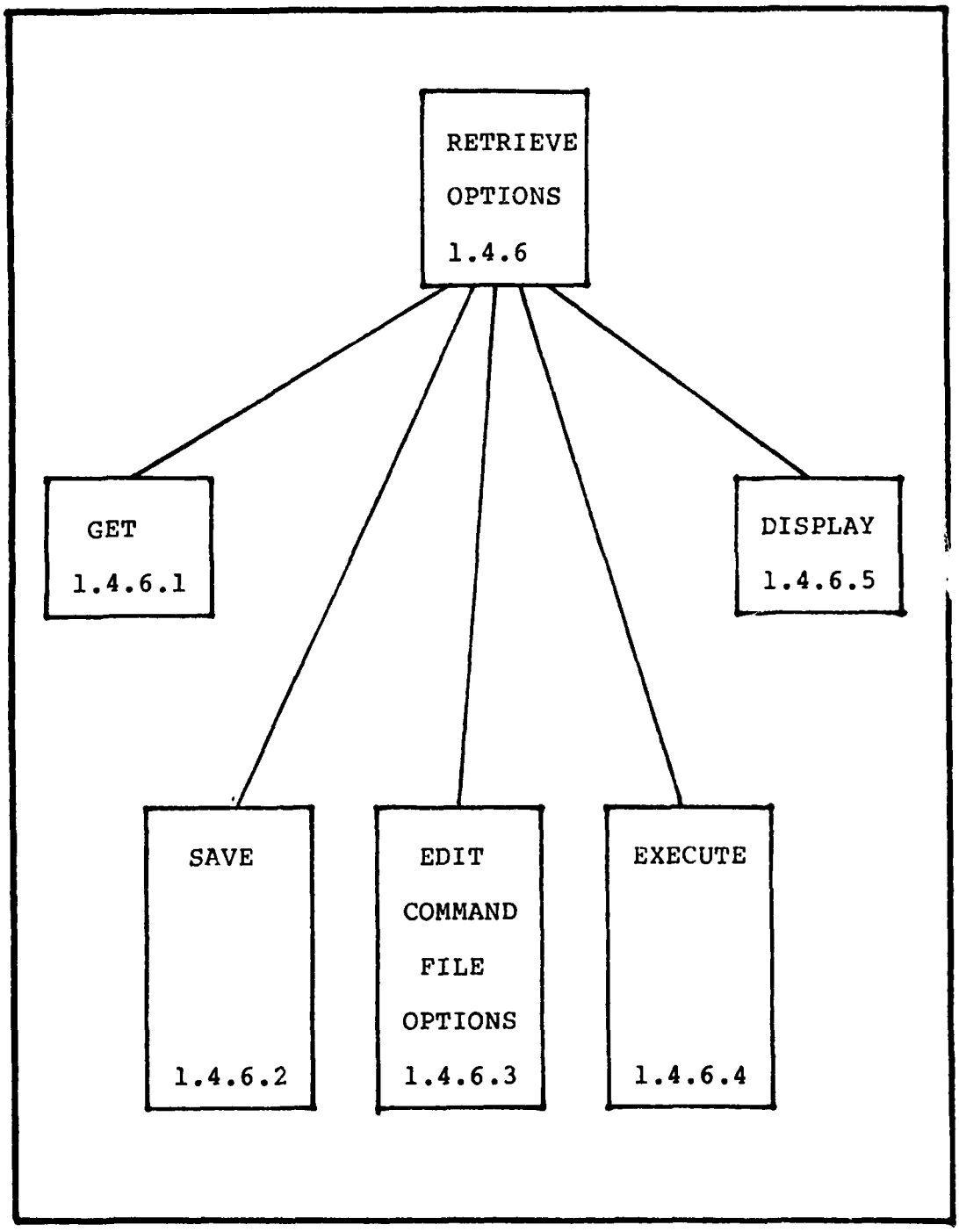


FIGURE C-20: RETRIEVE OPTION Module Level 1.4.6

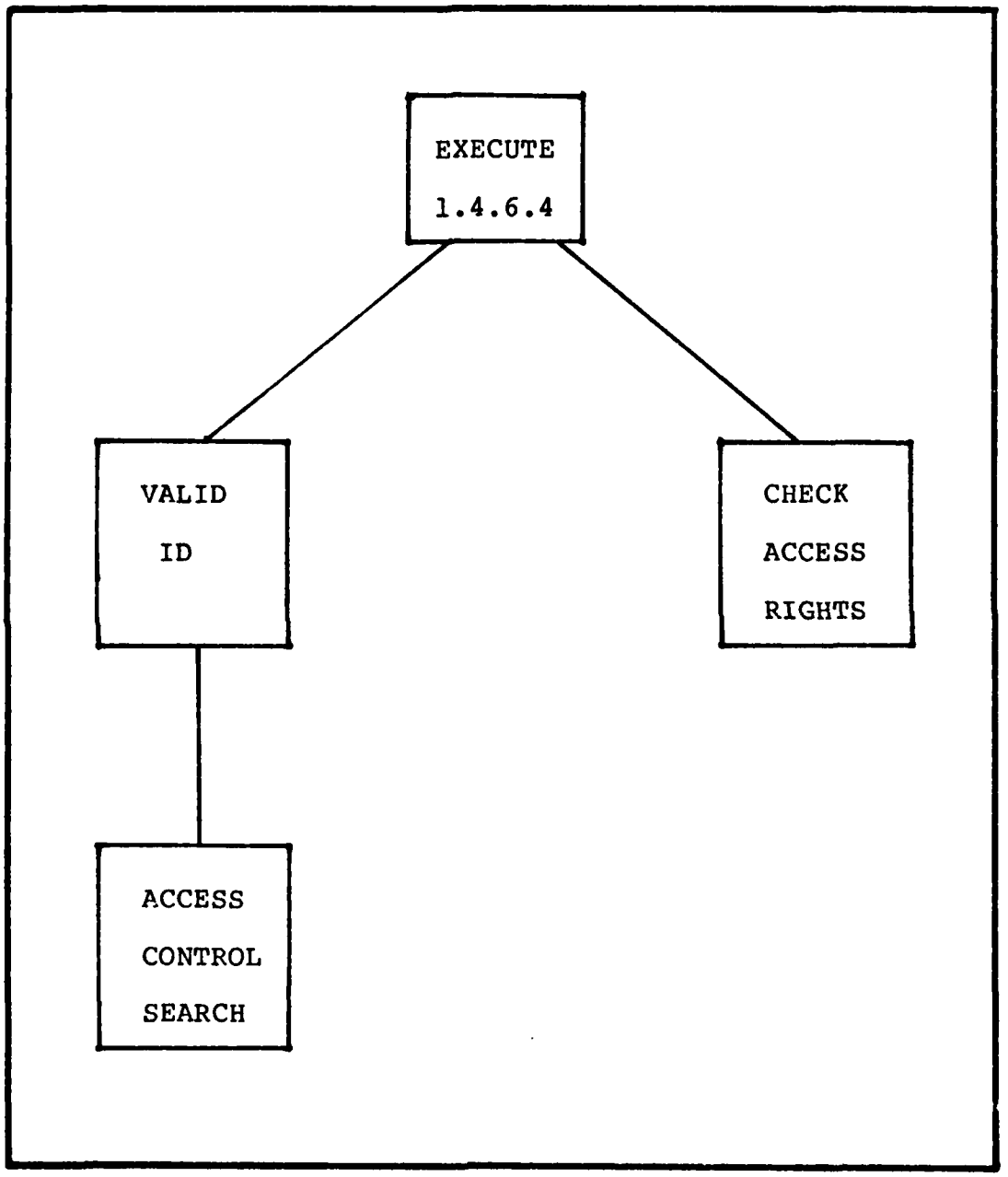


FIGURE C-21: EXECUTE Module Level 1.4.6.4

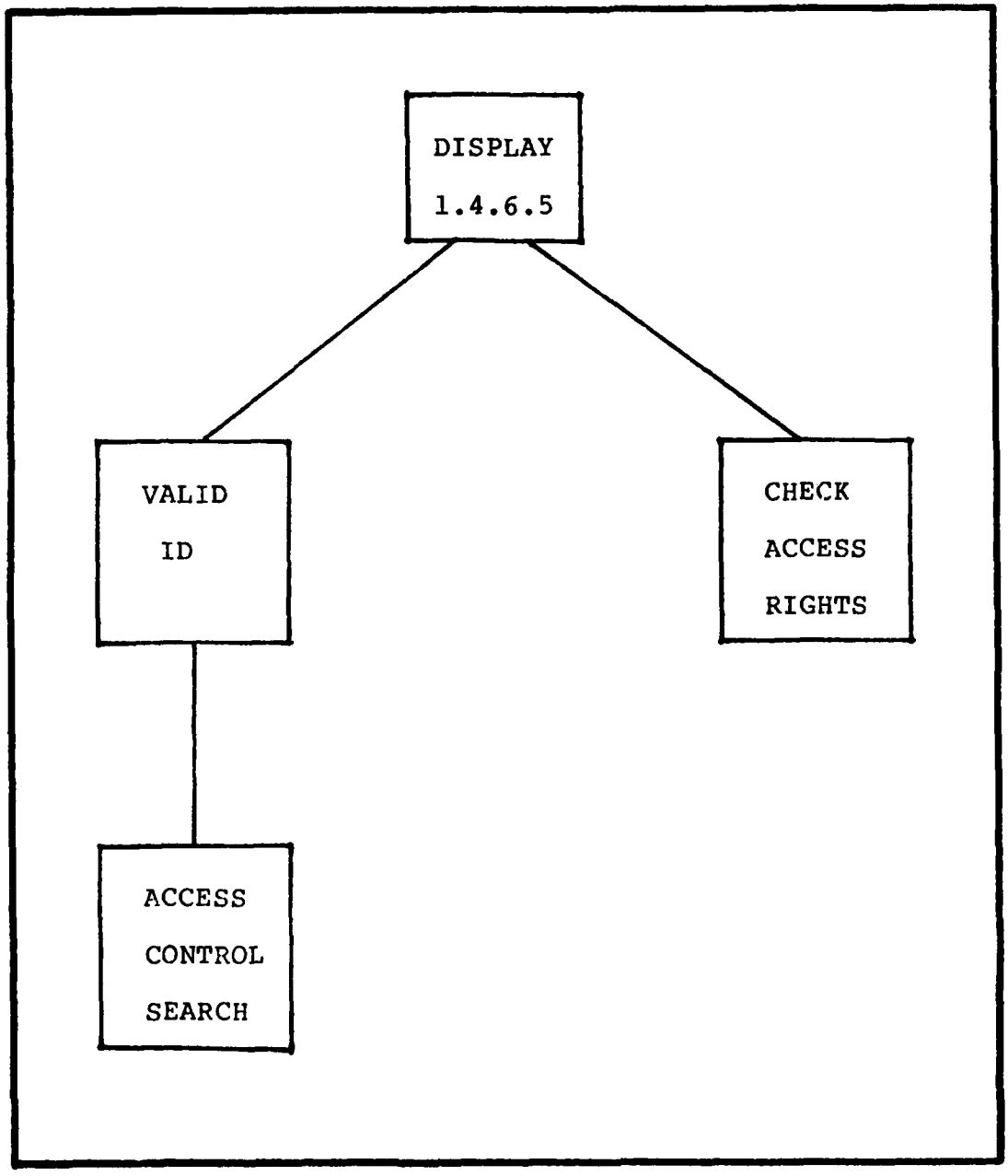


FIGURE C-22: DISPLAY Module Level 1.4.6.5

APPENDIX D

ARTICLE ON THE ANALYSIS AND DESIGN OF A COMPUTER SECURITY/RECOVERY SYSTEM FOR A RELATIONAL DATABASE MANAGEMENT SYSTEM

ABSTRACT

A Computer Security/Recovery System (CS/RS) model was designed with the goal of implementing it into a Relational Database Management System. In addition, the relational database was to serve as a pedagogical tool for teaching database management and manipulation techniques as well as security measures and issues.

Toward these goals, an extensive literature search and analysis of the current state of the art Computer Security models was required in order to identify current unresolved issues and areas of research. The advantages and disadvantages associated with six computer security models were explored and addressed in this thesis. Then the requirements for a CS/RS model were identified and defined, and based on certain favorable criteria, one of the computer security models analyzed was selected.

Further, only one of the six models had ever been implemented using a relational database and none had ever been implemented using a microcomputer. Consequently, the ability to take one of the existing computer security models and to apply the security concepts used within this

model was not an easy task and in some instances was not possible.

Finally, the CS/RS model designed did not include the hierarchally ordered military security sensitivity levels.

INTRODUCTION

The rising abuse rate of computers and the increasing threat to personal privacy through databases have stimulated an increased interest in the technical safeguards for data. Efforts to build secure computer systems have been underway for more than a decade. Many designs have been proposed, some prototypes have been constructed, and a few systems are approaching the production stage. A small number of systems in the Department of Defense (DOD) are even operating in "multilevel" mode: some information in any of these systems may have a classification higher than the clearance of some users (Ref 1:247).

Nearly all of the projects to design or construct secure systems for processing information have had a formal mathematical model for security as part of the top-down design definition of the system. The model functions as a concise and precise description of the behavior desired of the security-relevant portions of the system.

In order to build a secure system, designers need to first decide exactly what "secure" means for their particular needs. For instance, in a private company,

security may be related to nondisclosure of confidential accounting data or trade secrets, or to the enforcement of privacy regulations regarding personal medical or credit records. On the other hand, if national security data are involved, security becomes the protection of classified material as outlined in various DOD instructions and regulations. In either case computer security encompasses the measures required to (1) protect a computer-based system, including its physical hardware, personnel, and data against deliberate or accidental damage; (2) protect the system against denial of use by its rightful owners; and (3) protect information or data against divulgence to unauthorized recipients (Ref 2:55).

Most of the time it is evident that different security measures are required when limiting access to programs and data in a computer system having multiple users. In many systems where multiple users do exist there may be applications which require that not all users have the same idential authority. When this is true, it might be necessary to devise some scheme to ensure that the computer system prevents any unauthorized access to the data. For example, an instructor who maintains his course grades on an interactive system may wish to allow each student to read his own grades but not the grades of other students. He may also wish to allow any student to examine a histogram of the class grades for any assignment. In addition, the instructor would like an assistant to enter

new grades, but he wants to guarantee that each grade entered cannot be changed later without the instructor's specific approval.

Many other examples of systems requiring protection of information are encountered every day: credit bureau data banks; law enforcement information systems; time-sharing service bureaus; on-line medical information systems; and government social service data processing systems (Ref 2:67). These examples span a wide range of needs for organizational and personal privacy. All have in common controlled sharing of information among multiple users. All, therefore, require some type of security plan to ensure that unauthorized access, modification and/or deletion of data does not occur.

In regards to the Air Force Institute of Technology (AFIT) Digital Engineering Laboratory (DEL), most of the available computer operating systems do not address all areas of security. For instance, the security measures established by the DEL for the Digital Equipment Corporation (DEC) LSI-11 microprocessor configuration are quite lenient. As a result, any individual can access any one of the several operating system diskettes available for use on the LSI-11s. Each diskette has a system directory where the names of the files that have been created through the system are maintained. In addition, each operating system allows any user to observe its directory of file names. Consequently, any individual has the capability to

delete, modify, add, or rename any file within the operating system's directory. It is quite obvious that potential security problems may arise in this area.

STATEMENT OF PROBLEM

The problem addressed in this thesis is divided into three subtasks: first, an extensive literature search of the current state of the art Computer Security/Recovery Systems (CS/RSs) is required in order to identify current unresolved issues and areas of research; second, the requirements necessary for the design of a CS/RS model are identified and defined; and finally, a CS/RS model is designed for a microprocessor using a Relational Database Management System.

SCOPE

The major scope of this thesis consisted of an extensive literature search on each of the following computer security models and their associated advantages and disadvantages: (1) access matrix; (2) the University of California, Los Angeles (UCLA) Data Secure Unix; (3) take-grant; (4) high-water mark; (5) Bell and LaPadula (original); (6) Bell and LaPadula (revised); and (7) others as encountered. The literature search included but did not specifically focus on security measures in the following computer areas: (1) procedural; (2) personnel; (3) physical; (4) hardware; and (5) software.

The CS/RS design was based on several of the above

computer security models . In addition, any security measures pertaining to the LSI-11 operating systems and their peripherals was not discussed or implemented in the CS/RS model designed. Also, only suggestions for backup and recovery procedures are mentioned.

The procedures are given for the integration and testing of the CS/RS on the LSI-11 microprocessor using the Roth Relational Database System (Ref 6) and the UCSD Pascal Operating System (Ref 5).

SUMMARY OF MODELS

Figure D-1 compares the models with respect to approach, view of security, and use. To be useful, a comparison of the models should examine both the definition of security and the feasibility of implementing a computer system that performs the application required of it and can be verified to simulate the model. Unfortunately, such a comparison is diffcult because few implementations based on these models have reached a stage where verification of their security properties can be attempted (Ref 1:272). However, some of the models prove to be betterr candidates as bases for future secure computer systems than others.

Each model defines its own world and its own concept of security in that world. Thus a computer system that truly simulates any of the models will be secure in the sense defined by the model.

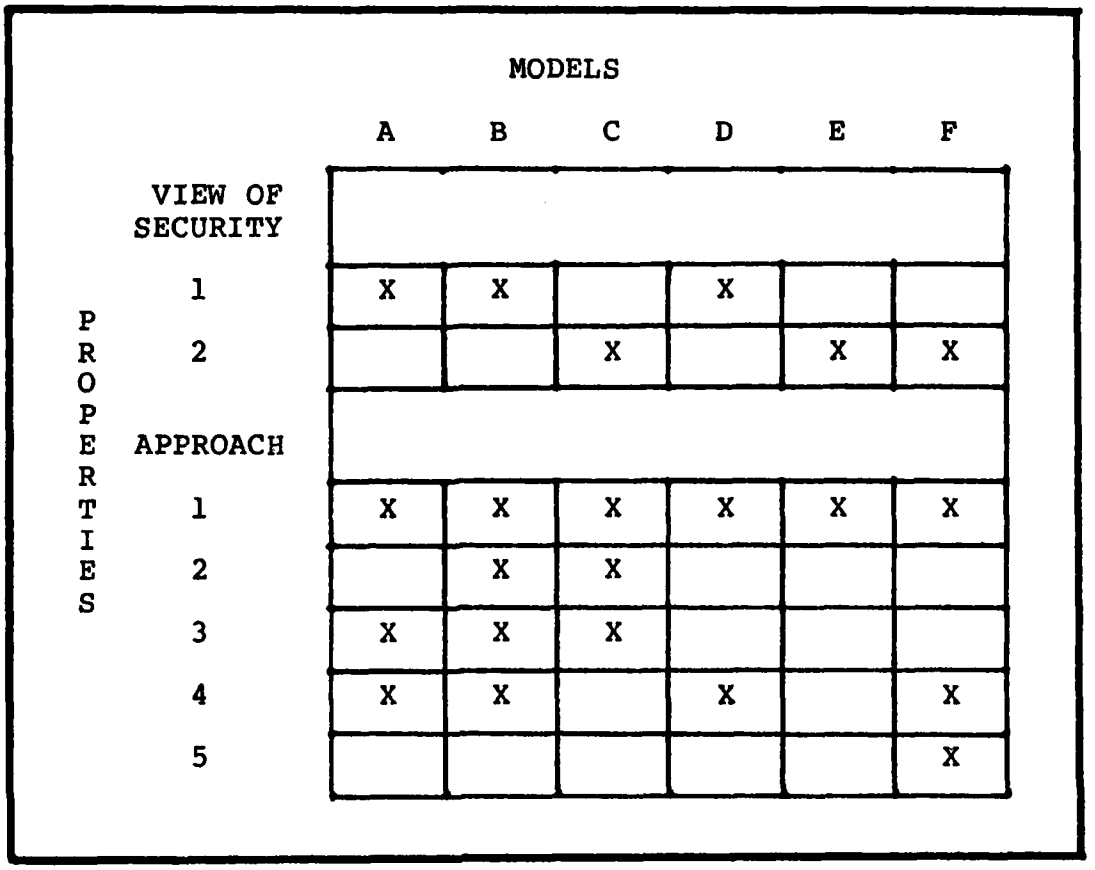


Figure D-1: Comparison of Properties of Models

MODELS:
- A = Access Matrix
- B = UCLA Date Secure Unix
- C = Take-Grant
- D = High-Water Mark
- E = Bell-LaPadula (original)
- F = Bell-LaPadula (revised)

PROPERTIES:

View of Security;
- 1 = Models access to objects without regard to contents
- 2 = Models flow of information among objects

Approach;
- 1 = Model focuses on system structure (files, processes)
- 2 = Model focuses on operations on capabilities
- 3 = Model separates protection mechanism and security policy
- 4 = Systems based on or represented by this model have been implemented
- 5 = Model has been implemented using a relational database

A problem common to all of the models is that they

define security as absolute (an operation is either secure or not secure) (Ref 1:273). This approach does not help the designer or the implementer who must make trade-offs between security and performance.

With respect to their definitions of security, the models can be divided into two groups: those that are concerned only with controlling direct access to particular objects (Access Matrix model, the UCLA Data Secure Unix model, and the Take-Grant model); and those that are concerned with information flows among objects assigned to security classes (original and revised Bell-LaPadula models). The High-Water Mark model falls between the two groups, since it is concerned with the security levels of the objects a process touches over time and only controls direct accesses to objects (Ref 1:273). Therefore, the appropriateness of a particular model will naturally depend on the application for which it is to be used.

MODEL SELECTION

Based on the requirements defined by the user a CS/RS model was selected. By observing Figure D-1 it appeared that any one of the six models would be appropriate. In addition, as previously mentioned the models could be divided into two groups: those that are concerned only with controlling direct access to particular objects (Access Matrix model, the UCLA Data Secure Unix, and the Take-Grant model); and those that are

concerned with information flows among objects assigned to security classes (original and revised Bell-LaPadula models). Since several of the requirements suggested protecting the flow of information from users or classes of users (objects) that are unauthorized to access data within a database, the second group of models seemed more likely to meet these requirements. In addition, the appropriateness of any particular model depended on the application for which it was to be used. For purposes of multilevel security classification levels, those in the first group required the addition of security classification policies and the assessment of indirect information flows (for example, timing and storage channels) at the implementation level. However, the second group of models already met this requirement.

The formal verification of properties of system designs was a very important issue and an active research topic. Consequently, only the Bell-LaPadula models have been applied in more than one formally specified system. In regards to the High-Water-Mark, Access Matrix, and Take-Grant models the properties associated with each of these models are not stated in a form suitable for automated verification.

Another important concept relating to the Bell-LaPadula (revised) model is that it was the only computer security model that had been implemented using a relational database. Thus, many of the properties

characterized by this model can be modified, if need be, and implemented into the design of the CS/RS model. In addition, most of the other models (Access Matrix, UCLA Data Secure Unix, and Take-Grant) separate the protection mechanisms and the security policies. However, Bell and LaPadula in their revised model demonstrated that this was not necessary because integrity and security could be defined to be dual in nature, thus avoiding any need for separate protection mechanisms and security policies.

Consequently, the CS/RS model designed was patterned after the revised Bell-LaPadula model. This was based on the following: the requirements defined by the user; the flexibility offered by the Bell- LaPadula revised model; and the fact that it was the only model that has been implemented using a relational database. However, many of the techniques used in the revised model can not be implemented because of the constraints imposed on the design by the LSI-11 and the Roth relational database. These constraints as well as other modifications needed for the CS/RS design are discussed in detail later in Appendix.

CS/RS MODEL

The original relational DBMS was conceived by Mark A. Roth (Ref 6) in a 1979 thesis. Then in 1982, Linda M. Rogers (Ref 34) separated the DBMS into the Data Definition Facility and the Roth Relational Database System. In addition, executable baseline models were

developed for each. These baseline models were used as the basis for implementing the CS/RS model designed. Figure D-2 represents the Data Definition Facility structure chart with any new or modified modules represented by an "*". Figure D-3 represents the Roth Relational Database System structure chart with any new or modified modules represented by an "*".

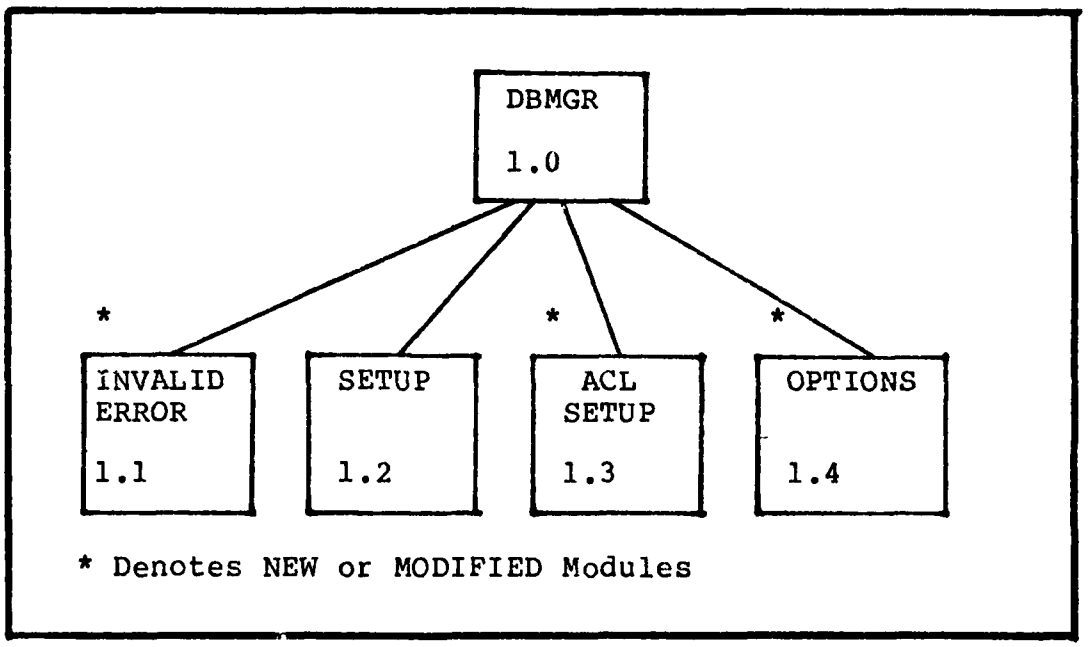


FIGURE D-2: Data Definition Facility Structure Chart

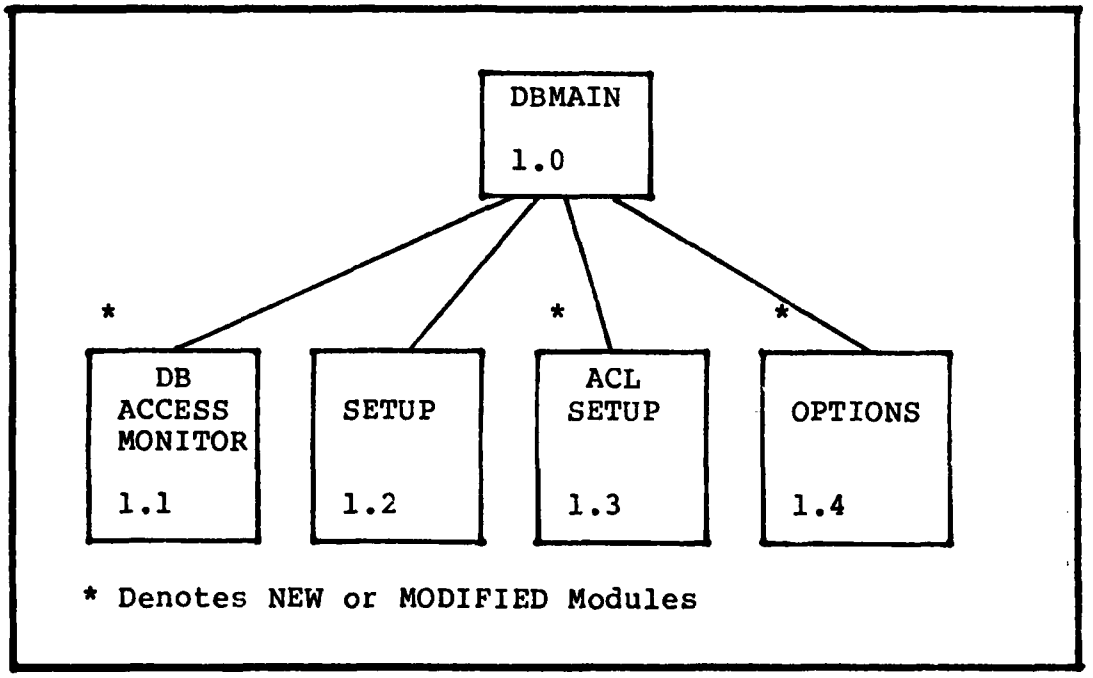


FIGURE D-3: Roth Relational Database Structure Chart

Due to time constraints, none of the CS/RS modules were tested. However, all the CS/RS modules designed were coded, implemented, and shown to be free from compile errors.

The CS/RS model was divided into the following three areas: the DBMS Access Monitor; the Relation Access Control List; and the Relation Access Monitor.

The DBMS Access Monitor was designed to prevent unauthorized users from gaining access to the DBMS. This was accomplished by assigning authorized users a unique user-id and password which would be used for logging onto the DBMS. Once the user-id and password had been entered,

they would be passed by the software to the DBMS Access Monitor. Then the DBMS Access Monitor would verify the logon attempt by comparing the user-id and password entered to the tuples contained in a permanent relation called ACCESS. These tuples would consist of an ID, password, user-name, and classification level. In addition, only those users having access to the DBMS would be contained in this relation. If a match was found then the user would be granted permission to access the DBMS; otherwise permission would be denied.

The Relation Access Control List was designed as a binary tree for storing the following information: the relation names defined in the DDL; the owner-ids; and the lists of user-ids with their assigned access rights. In addition, the owner of a relation would be responsible for entering this information.

In order to enter information into the Relational Access Control List the owner of a relation would go through the normal logon procedures. Once permission to access the DBMS had been granted the owner would call up the Relation Access Control List module. Then the owner would enter the relation name and a unique, owner defined owner-id. After this information had been entered the owner would need to enter the user's name (first name, blank, middle initial, blank, last name) and the access rights (read, insert, delete, modify, or execute) granted to this user. It was necessary to enter the user's name

in order to prevent any compromising of user-ids. Once the user's name and access rights had been entered, the user's name would be passed by the software to the DBMS Access Monitor which would retrieve the tuple from the ACCESS relation matching this user's name. If the tuple was found then the user-id would be retrieved and passed back by the software to the Relation Access Control List module. Then the user-id and access rights granted to this user would be entered into the list of users with their associated access rights. It is important to note that only the user-ids and access rights may be added, deleted, or modified in the Relation Access Control List.

The Relation Access Monitor was designed for verifying whether a user had been granted access (read, insert, delete, modify, or execute) to a relation. If access had been granted to a relation then each attribute's classification level would be checked to ensure that the user's classification level (which was retrieved from the ACCESS relation at logon) was greater than or equal to the classification level of the attribute. If not, access to this attribute would be denied. It is important to note that classification levels were only assigned to the attributes and not to the relation or to the tuples.

CS/RS MODEL CONSTRAINTS

Several constraints were imposed on the CS/RS model by the LSI-11 and the Roth relational database. For

example, many of the security measures implemented in the Bell-LaPadula (revised) model were enforced at the operating system level. However, the UCSD Pascal Operating System did not provide this luxury. Consequently, many of the following security measures could not be implemented:

1) A set of directory functions which is used by tne Bell-LaPadula model in searching for directories and for objects;

2) A set of access modes consisting solely of observe and modify;

3) The implementation of integrity levels to ensure any additions, modifications, or deletions to a relation do not jeopardize other relations in the DBMS; and

4) The use of the direct access function for enforcing the protection requirements.

A second and very important constraint was that the available core memory on the LSI-11 was too limiting. Since most of the Roth relational database code had previously been segmented, it was oftentimes difficult to work in the necessary CS/RS modules without resegmenting. Another LSI-11 constraint involved backup and recovery procedures. Currently, this is almost impossible to do under the UCSD Pascal Operating System. However, backup and recovery procedures can be written to interface with the Roth Relation Database System. These procedures should be performed automatically by the DBMS without user intervention. Finally, compiling and linking of the binary code file into an executable module took an enormous amount of time, especially when segmentation was

involved.

SUMMARY

The title of this thesis implies that two goals were sought. One was to analyze several computer security models and the other was to design a CS/RS model for a Relational Database Management System. The first goal was met, however it was quite time consuming. A decision was made to summarize the computer security models and to list the advantages and disadvantages associated with each.

The second goal was achieved in that a CS/RS model was designed from those security models analyzed for the Roth Relational Database System. However, the integration and testing of this model has yet to be accomplished. Since the Roth Relational Database was to be used in a pedagogical environment, the security measures implemented in the model will prove to be very beneficial. In addition, a tremendous amount of flexibility was available in selecting the actual design of the CS/RS model. This was primarily due to the fact that very few of the computer security models analyzed had ever been implemented using a relational database.

FINAL COMMENT

Relational databases promise to provide a simple and flexible approach to a person or business's information retrieval needs. However, they require looking at security measures from a different perspective when compared to

other types of databases. Consequently, this thesis effort was an attempt to take some of those security measures and design them into a viable CS/RS model that could be used in a Relational Database Management System.

VITA

William Reed was born December 28,1951 at Norfolk, Nebraska. He attended Randolph Public High School at Randolph, Nebraska, graduating May 1970. He attended the University of Nebraska at Lincoln, graduating in May 1974, with a B. S. in Math Education. Upon graduation he accepted a two-year ROTC Scholarship to attend the University of Nebraska at Lincoln, graduating in May 1976, with a M. S. in Psychology.

He came on active duty in September 1976, and was assigned to the Computer Services Squadron at Offutt AFB, Nebraska. After four and one-half years at Offutt, he was selected to attend the Air Force Institute of Technology. On December 17, 1982 he graduated with an M. S. (Information Systems).

Permanent Address: Wausa, NE 68786.